UNIVERSITY OF CALIFORNIA SAN DIEGO

Symmetric Cryptography: New Definitions and Schemes

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Computer Science

by

Ruth Ii-Yung Ng

Committee in charge:

Professor Mihir Bellare, Chair
Professor David Cash
Professor Alin Deutsch
Professor Farinaz Koushanfar
Professor Daniele Micciancio
Professor Deian Stefan

2021

The Dissertation of Ruth Ii-Yung Ng is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2021

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I want to thank all those who contributed to my PhD in one way or another. There are so many people without whom I would not have made it here. Of these, I especially thank the following people:

My advisor, Mihir Bellare, for his support and guidance during this PhD process. I have the highest admiration for his critical thinking and attention to detail, and consider it a great honor to have been advised by him.

David Cash for his guidance in the projects we have done together. Through his leadership and bountiful patience for me, I have learned so much about how to be a better researcher.

Khoongming Khoo and Ariel Feldman who introduced me to research and aided me in getting to UCSD.

All who have researched with me these years: Mihir Bellare, David Cash, Francesca Falzon, Alexander Hoover, William Hoover, Adam Rivkin, Jiahao Sun and Björn Tackmann. Your ideas and hard work inspire me and I would not be here without you all.

The incredibly supportive computer science communities I have been a part of at UCSD and UChicago. In particular, I thank the graduate students I have talked research (and so much more) with all these years for their companionship and camaraderie.

DSO National Laboratories in Singapore which has financially supported the entirety of my PhD process.

My family and friends for their encouragement and love. Thanks especially to Patrick Chen who has been right there beside me this entire PhD journey.

Chapter 1, in full, is a reprint of the material as it appears in Advances in Cryptology – CRYPTO 2019. Bellare, Mihir; Ng, Ruth; Tackmann, Björn, Springer Lecture Notes in Computer Science volume 11692, 2019. The dissertation author was the primary investigator and author of this paper.

Chapter 2, in full, is a reprint of the material as it appears in International Conference on Applied Cryptography and Network Security – ACNS 2021. Cash, David; Ng, Ruth; Rivkin,

Adam, Springer Lecture Notes in Computer Science volume 12727, 2021. The dissertation author was the primary investigator and author of this paper.

Chapter 3, in full, is currently being prepared for publication of the material. Cash, David; Hoover, Alexander; Ng, Ruth. The dissertation author was the primary investigator and author of this paper.

VITA

2015	Bachelor of Arts, University of Chicago
2015	Bachelor of Science, University of Chicago
2018	Master of Science, University of California San Diego
2021	Doctor of Philosophy, University of California San Diego

ABSTRACT OF THE DISSERTATION

Symmetric Cryptography: New Definitions and Schemes

by

Ruth Ii-Yung Ng

Doctor of Philosophy in Computer Science

University of California San Diego, 2021

Professor Mihir Bellare, Chair

We say that cryptographic schemes are "symmetric" whenever the sender and receiver share the same key. In this work we consider and evaluate two forms of such: authenticated encryption (AE) and structured encryption (StE).

AE is used to encrypt messages in a way that guarantees the privacy and integrity of the data therein. Our work draws attention to a gap between the theory and usage of nonces with regard to how nonces are communicated from sender to receiver. We bridge this with a new *nonce-hiding* treatment of authenticated encryption and propose simple, efficient schemes that conform to these new definitions.

StE is used to encrypt a large database for storage on an untrusted server in such a way

that the client retains the ability to perform fast query-based search on the data. We first study the problem of indexing joins in encrypted SQL databases using StE. We introduce a new technique called *partially precomputed joins* which achieves lower leakage than existing techniques. We devise a *hybrid indexing* scheme which uses both indexes and provide a client-side leakage aware query optimization heuristic with which the client can choose which index to use at query time. We evaluate our indexing method and heuristic with simulations on real datasets.

We then revisit the idea of Chase and Kamara (ASIACRYPT 2010) to build more complex StE schemes from simple ones via the composition of dictionary and multimap StE. We show that the intuitive composition can run into some subtle issues related to the coordination of their simulators. We then address this situation in two ways. First, we provide a sufficient condition called *content obliviousness* under which this issue can be resolved. Second, we suggest an alternate *monolithic approach* that avoids composition altogether which uses a single data structure that supports more complex queries. To analyze this construction, we need a basic form of *key-dependent security* for StE, so we initiate a theoretical study of such by giving impossibility results, constructing generic transforms and evaluating existing schemes.

Introduction

Symmetric cryptography is used on a daily basis by almost everyone for securely transmitting or storing data. This includes the secure communication of web data when one accesses an HTTPS website or the secure storage of files using cloud storage systems.

More formally, a symmetric cryptography scheme will encrypt and decrypt data using the same key. These procedures may be called by different people or by the same person. An example of the former is when Alice sends Bob an encrypted file. In this case, Alice encrypts and Bob decrypts the file with a shared key. An example of the latter is when Alice is using a cloud storage service hosted by Bob. Here, Alice will encrypt and upload her data using a key not known to Bob. She will later download her data and decrypt it using this same key.

In this work, our goal is to model real-world use-cases of symmetric cryptography. In each case, we use the same general process which starts with defining a formalism within the widely-used game-playing framework of Bellare and Rogaway [30]. This consists of a syntax and a security notion. Syntax provides the "nuts and bolts" of the primitive by detailing the algorithms within the model. This includes their inputs and outputs, access to randomness, correctness conditions and more. The security notion captures an intuitive goal which usually mirrors desirable security properties in the real world. The next step is to actualize these formalisms with cryptographic schemes. These schemes define algorithms in accordance to the syntax and provably achieve the desired security notions outlined by the formalism. Finally, one can evaluate and compare the security and efficiency of these schemes either experimentally (e.g. simulations on real data) or theoretically (e.g. asymptotic bounds).

Nonce-based authenticated encryption.

The model address first is a simple one: Alice encrypts a message of her choice and sends it to Bob over an insecure channel. This encryption scheme should protect both the privacy and integrity of the message until Bob decrypts it (i.e. it cannot be read or tampered with by an attacker). Our formalism is an extension of Rogaway's nonce-based notion [120, 122] where encryption is deterministic but also takes a non-repeating quantity called a nonce.

In Rogaway's syntax, which we call NBE1, decryption also receives the nonce as an input which gives rise to a gap between the theory and usage of nonce-based encryption surrounding how this nonce should be communicated to Bob. The theory (NBE1) assumes that this nonce transmission "out-of-band" [122] and therefore "outside of the model" [120]. However, usage cannot so dismiss it and often send this nonce in the clear along with the ciphertext to Bob. We demonstrate that the latter approach can compromise data privacy and propose a modified syntax, called NBE2, where nonce communication is incorporated into the formalism. We conclude this work with a portfolio of transforms which build NBE2 schemes from NBE1 schemes. This enables the existing systems to cheaply "upgrade" the cryptography they use and avoid the danger posed by sending nonces in the clear.

Join indexing in encrypted SQL systems.

In our second model, Alice is the administrator of a SQL database and Bob is an honest-but-curious cloud service provider. The syntax allows Alice to encrypt and store her database on Bob's server, then later makes select and join SQL queries to this data via query-specific tokens. We use a semantic security notion in this formalism to allow for efficient symmetric constructions where Bob is allowed some controlled leakage on the database. More specifically, the security notion is parametrized using a leakage profile which upper-bounds what Bob can learn about the data from Alice's encrypted database and query tokens.

Our work introduces a new (secure) indexing technique which we call partially precomputed (PP) joins. This is an extension of Kamara and Moataz's fully precomputed join indexing

[89]. We show that when Alice issues join queries of the form "**select** $*$ **from** T_1 **join** T_2 **on** $at_1 = at_2$", PP indexing incurs strictly less leakage and bandwidth, meaning it has superior security and efficiency compared to FP indexing.

We then incorporate PP joins into the state-of-the-art indexing schemes thereby broadening our query support to allow recursion, selections and cluster joins. In doing so, we notice that while PP indexing still results in a scheme with strictly less leakage, it has significantly worse bandwidth on certain classes of complex queries.

To address this, we propose a hybrid indexing scheme where Alice stores both indexes on Bob's server and decides, at query time, which she will use. We then provide such a heuristic which helps Alice select a query execution plan that offers her the best-possible security without exceeding a predetermined bandwidth budget for each query.

We conclude our work with simulations on real-world datasets. These back-up our comparisons between FP and PP indexing and demonstrate the effectiveness of our heuristic.

Composition and key-dependent structured encryption.

Our third model is also in the client-server model but with a slightly modified syntax: Alice's data is now a data structure with "double-leveled indexing". An example of such is an encrypted file system which allows document retrieval via keywords. This can be indexed using two dictionaries – the first associates a document identifier to each file's contents while the second maps each keyword to the list of relevant document identifiers. To store this index on the server, a natural approach would be to use two dictionary encryption schemes (also known as Searchable Symmetric Encryption schemes), composed in the manner proposed by Chase and Kamara [52].

In our work, we demonstrate that standard semantic security for the primitive dictionary encryption schemes does not enable some straightforward reductions that may appear to work at first glance, and identify some steps in prior work that exhibit this gap. We then address the double-level indexing problem in two ways. First, we give an extra condition on the primitives'

leakage profiles which we call content obliviousness and show that the prior proof approach can be recovered using this. Our second approach is to give a monolithic solution to the problem which pre-processes data into a single data structure and employ a single dictionary encryption primitive. This approach has strictly better security than using CK's composition technique using standard primitives.

In analyzing the security of this monolithic approach, another proof challenge comes: we need to store key-dependent (KD) material in the data structure. Such issues have arisen in other forms of encryption, and have been well studied under various notions of KD message security. We adapt this line of thinking and give a new KD notion of semantic security and show that many state-of-the-art dictionary encryption primitives achieve it. We then show that such scheme suffice to render the monolithic solution secure with no additional assumptions.

We believe a broader variant of our KD security notion may be of independent interest and therefore provide a set of foundational results on this. In doing so, we encounter some subtle issues which do not appear in prior KD security notions.

Chapter 1

Nonces are Noticed: AEAD Revisited

1.1 Introduction

This paper revisits nonce-based symmetric encryption, raising some concerns, and then addressing them, via a new syntax, a new framework of security definitions, and schemes that offer both usability and security benefits.

Background.

As the applications and usage of symmetric encryption have evolved and grown, so has a theory that seeks to support and guide them. A definition of symmetric encryption (as with any other primitive) involves a *syntax* and then, for this syntax, definitions of *security*. In the first modern treatment [24], the syntax asked the encryption algorithm to be randomized or stateful. Security for these syntaxes evolved from asking for various forms of privacy [24] to asking for both privacy and authenticity [29, 26, 96], inaugurating authenticated encryption (AE). The idea that encryption be a deterministic algorithm taking as additional input a non-repeating quantity called a nonce seems to originate in [124] and reached its current form with Rogaway [120, 122].

NBE1 and AE1-security.

We refer to the syntax of this current form of nonce-based symmetric encryption [120, 122] as NBE1. An NBE1 scheme $\mathsf{SE1}$ specifies a *deterministic* encryption algorithm $\mathsf{SE1.Enc}$ that takes the key K, a nonce N, message M and a header (also called associated data) H to return

what we call a core ciphertext C_1. Deterministic decryption algorithm $\mathsf{SE1.Dec}$ takes K, N, C_1, H to return either a message or $\perp$.

Security asks for privacy of M and integrity of both M and H *as long as nonces are unique*, meaning not re-used. Rogaway's formalization [120] asks that an adversary given oracles for encryption (taking nonce, message and header) and decryption (taking nonce, core ciphertext and header) be unable to distinguish between the case where they perform their prescribed tasks under a hidden key, and the case where the former returns random strings and the latter returns $\perp$, as long as the adversary does not repeat a nonce across its encryption queries. We will refer to this as basic AE1-security.

NBE1 providing basic AE1-security has been the goal of recent schemes, standards and proposed standards, as witnessed by GCM [105, 61] (used in TLS), OCB [124, 121, 99], CAESAR candidates [33] and RFC 5116 [104]. The security of NBE1, which we revisit, is thus of some applied interest.

The gap.

Our concern is a gap between theory and usage that can result in privacy vulnerabilities in the latter. Recall that the decryption algorithm $\mathsf{SE1.Dec}$, to be run by the receiver, takes as input not just the key K, core ciphertext C_1 and header H, but *also the nonce* N. The theory says that how the receiver gets the nonce is "outside of the model" [120] or that it is assumed to be communicated "out-of-band" [122]. Usage cannot so dismiss it, and must find a way to convey the nonce to the receiver. The prevailing understanding, reflected in the following quote from RBBK [124], is that this is a simple matter— if the receiver does not already have the nonce N, just send it in the clear along with the core ciphertext C_1:

> The nonce N is needed both to encrypt and to decrypt. Typically it would be communicated, in the clear, along with the (core) ciphertext.

RFC 5116 is a draft standard for an interface for authenticated encryption [104]. It also considers it fine to send the nonce in the clear:

> ... there is no need to coordinate the details of the nonce format between the encrypter and the decrypter, as long *the entire nonce is sent* or stored with the ciphertext and is thus available to the decrypter ... the nonce MAY be stored *or transported* with the ciphertext

To repeat and summarize, the literature and proposed standards suggest transmitting what we call the "full" ciphertext, consisting of the nonce and the core ciphertext. Yet, as we now explain, this can be wrong.

Nonces can compromise privacy.

We point out that communicating a nonce in the clear with the ciphertext can damage, or even destroy, message privacy. One simple example is a nonce $N = F(M)$ that is a hash —under some public, collision-resistant hash function F— of a low-entropy message M, meaning one, like a password, which the attacker knows is likely to fall in some small set or dictionary D. Given a (full) ciphertext $C_2 = (N, C_1)$ consisting of the core ciphertext $C_1 = \mathsf{SE1.Enc}(K, N, M, H)$ together with the nonce $N = F(M)$, the attacker can recover M via "For $M' \in D$ do: If $F(M') = N$ then return M'." To take a more extreme case, consider that the nonce is some part of the message, or even the entire message, in which case the full ciphertext clearly reveals information about the message.

The concern that (adversary-visible) nonces compromise privacy, once identified, goes much further. Nonces are effectively meta-data. Even recommended and innocuous-seeming choices like counters, device identities, disk-sector numbers or packet headers reveal information about the system and identity of the sender. For example, the claim that basic-AE1-secure NBE1 provides anonymity —according to [123, Slide 19/40], this is a dividend of the requirement that core ciphertexts be indistinguishable from random strings— is moot when the nonce includes sender identity. Yet the latter is not only possible but explicitly recommended in RFC 5116 [104], which says: "When there are multiple devices performing encryption ... use a nonce format that contains a field that is distinct for each one of the devices." As another concrete example, counters are *not* a good choice of nonce from a user privacy perspective, as pointed out by

Bernstein [32] and the ECRYPT-CSA *Challenges in Authenticated Encryption* report [14].

The above issues apply to all NBE1 schemes and do not contradict their (often, proven) AE1-security. They are not excluded by the unique nonce requirement or by asking for misuse resistance [125], arising in particular for the encryption of a single message with a single corresponding nonce.

A natural critique is that the privacy losses we have illustrated occur only for "pathological" choices of nonces, and choices made in practice, such as random numbers or counters, are "fine." This fails, first, to recognize the definitional gap that allows the "pathological" choices. With regard to usage, part of the selling point of NBE1 was exactly that *any* (non-repeating, unique) nonce is fine, and neither existing formalisms [120] nor existing standards [104] preclude nonce choices of the "pathological" type. Also, application designers and users cannot, and should not, carry the burden of deciding which nonces are "pathological" and which are "fine," a decision that may not be easy. (And as discussed above, for example, counters may *not* be fine.) Finally, Section 1.9 indicates that poor choices can in fact arise in practice.

Our perspective is that the above issues reflect a gap between the NBE1 formalism and the privacy provided by NBE1 in usage. Having pointed out this gap, we will also bridge it.

Contributions in brief.

The first contribution of this paper is to suggest that the way NBE1 treats nonces can result (as explained above) in compromise of privacy of messages or users. The second contribution is to address these concerns. We give a modified syntax for nonce-based encryption, called NBE2, in which decryption does not get the nonce, a corresponding framework of security definitions called AE2 that guarantee nonce privacy in addition to authenticity and message privacy, and simple ways to turn NBE1 AE1-secure schemes into NBE2 AE2-secure schemes.

AE2-secure NBE2 obviates application designers and users from the need to worry about privacy implications of their nonce choices, simplifying design and usage. With AE2-secure NBE2, one can use any nonce, even a message-dependent one such as a hash of the message,

without compromising privacy of the message. And the nonces themselves are hidden just as well as messages, so user-identifying information in nonces doesn't actually identify users.

Our NBE2 syntax.

In an NBE2 scheme $\mathsf{SE2}$, the inputs to the deterministic encryption algorithm $\mathsf{SE2.Enc}$ continue to be key K, nonce N, message M and header H, the output C_2 now called a ciphertext rather than a core ciphertext. The deterministic decryption algorithm $\mathsf{SE2.Dec}$ *no longer gets a nonce*, taking just key K, ciphertext C_2 and header H to return either a message M or $\perp$.

Just as an interface, NBE2 already benefits application designers and users, absolving them of the burden they had, under NBE1, of figuring out and architecting a way to communicate the nonce from sender to receiver. The NBE2 receiver, in fact, is nonce-oblivious, not needing to care, or even know, that something called a nonce was used by the sender. By reducing choice (how to communicate the nonce), NBE2 reduces error and misuse.

We associate to a given NBE1 scheme $\mathsf{SE1}$ the NBE2 scheme $\mathsf{SE2} = \mathbf{TN}[\mathsf{SE1}]$ that sets the ciphertext to the nonce plus the core ciphertext: $\mathsf{SE2.Enc}(K,N,M,H) = (N, \mathsf{SE1.Enc}(K,N,M,H))$ and $\mathsf{SE2.Dec}(K,(N,C_1),H) = \mathsf{SE1.Dec}(K,N,C_1,H)$. We refer to **TN** as the Transmit Nonce transform. This is worth defining because it will allow us, in Section 1.5, to formalize the above-discussed usage weaknesses in NBE1, but $\mathsf{SE2} = \mathbf{TN}[\mathsf{SE1}]$ is certainly not nonce hiding and will fail to meet the definitions we discuss next.

Our AE2-security framework.

Our AE2 game gives the adversary an encryption oracle ENC (taking nonce N, message M and header H to return a ciphertext C_2) and decryption oracle DEC (as per the NBE2 syntax, taking ciphertext C_2 and header H but no nonce, to return either a message M or $\perp$). When the challenge bit is $b = 1$, these oracles reply as per the encryption algorithm $\mathsf{SE2.Enc}$ and decryption algorithm $\mathsf{SE2.Dec}$ of the scheme, respectively, using a key chosen by the game. When the challenge bit is $b = 0$, oracle ENC returns a ciphertext that is drawn at random from a space

$\mathsf{SE2.CS}(|N|,|M|,|H|)$ that is prescribed by the scheme SE2 and that depends only on the lengths of the nonce, message and header, which guarantees privacy of both the nonce and message. (This space may be, but unlike for AE1 need not be, the set of all strings of some length, because NBE2 ciphertexts, unlike NBE1 core ciphertexts, may have some structure.) In the $b = 0$ case, decryption oracle DEC returns $\perp$ on any non-trivial query. The adversary eventually outputs a guess b' as to the value of b, and its advantage is $2\Pr[b = b'] - 1$.

We say that SE2 is AE2$[\mathcal{A}]$-secure if practical adversaries in the class $\mathcal{A}$ have low advantage. Let $\mathcal{A}^{\mathrm{ae2}}_{\mathrm{u\text{-}n}}$ be the class of unique-nonce adversaries, meaning ones that do not reuse a nonce across their ENC queries. We refer to AE2$[\mathcal{A}^{\mathrm{ae2}}_{\mathrm{u\text{-}n}}]$-security as basic AE2-security. As the nonce-hiding analogue of basic AE1-security, it will be our first and foremost target.

Before moving to schemes, we make two remarks. First that above, for simplicity, we described our definitions in the single-user setting, but the definitions and results in the body of the paper are in the multi-user setting. Second, the framework of a single game with different notions captured via different adversary classes allows us to unify, and compactly present, many variant definitions, including basic, advanced (misuse resistance), privacy-only and random-nonce security, and in Section 1.3 we give such a framework not just for AE2 but also for AE1.

Our general results.

The analysis of schemes is simplified by some general results we give in Section 1.4. Foremost is a decomposition theorem that tightly bounds the ae-advantage of a given adversary in terms of the advantage of a privacy-only adversary (no decryption queries) and a very restricted type of authenticity adversary that we call *orderly*— it needs only verification queries (not decryption queries) and these follow its encryption queries and are all made in parallel. Here we are following Bose, Hoang and Tessaro (BHT) [39], who gave such a result for basic AE1-security. Theorem 1 slightly improves their bound and also extends the result to both advanced security and AE2, our single theorem thus capturing four results. Additionally, Theorem 2 states

the standard reduction of mu security to su security and Theorem 3 reduces security for random nonces to security for unique nonces.

Our transforms.

In the presence of a portfolio of efficient AE1-secure NBE1 schemes supported by proofs of security with good concrete bounds [124, 105, 33, 99, 84, 133, 108, 75, 115, 74, 39, 82], designing AE2-secure NBE2 schemes from scratch seems a step backwards. Instead we give simple, cheap ways to transform AE1-secure NBE1 schemes into AE2-secure NBE2 schemes, obtaining a corresponding portfolio of AE2-secure NBE2 schemes and also allowing implementors to more easily upgrade deployed AE1-secure NBE1 to AE2-secure NBE2.

Since NBE2 schemes effectively take care of nonce communication, we expect ciphertext length to grow by at least $\mathsf{SE1.nl}$, the nonce length of the base NBE1 scheme. The *ciphertext overhead* is defined as the difference between the ciphertext length and the sum of plaintext length and $\mathsf{SE1.nl}$. *All our transforms have zero ciphertext overhead.* One challenge in achieving this is that nonce lengths like $\mathsf{SE1.nl} = 96$ are widely-used but short of the block length 128 of many blockciphers, precluding inclusion of an extra blockcipher output in the ciphertext. With regard to computational overhead, the challenge is that it should be constant, meaning independent of the lengths of the message and header for encryption, and of the ciphertext and header for decryption. *All our transforms have constant computational overhead.*

The following discussion first considers achieving basic security and then advanced security. Security attributes of our corresponding "Hide-Nonce (HN)" transforms are summarized in Figure 1.1. In the table $\mathsf{SE1}$ denotes an NBE1 scheme, F a PRF, E a block cipher, and TE a variable-length tweakable block cipher. Spl is a splitting function, and ℓ, ℓ_z are non-negative integer parameters. Note that the advanced security of **HN1** only holds when ciphertexts are sufficiently large (e.g. 128 bits), and **HN2**'s depends on the length of the stolen ciphertext.

Table 1.1. Security attributes of the NBE2 schemes defined by our HN transforms. A blank entry in the Basic column means the transform is not for that purpose. Note that the advanced security of **HN1** and **HN2** only hold under certain conditions.

NBE2 scheme	**AE2-security provided**	
	Basic	Advanced
HN1$[\mathsf{SE1}, \mathsf{F}]$	Yes	Yes
HN2$[\mathsf{SE1}, \ell, \mathsf{E}, \mathsf{Spl}]$	Yes	Yes if $\ell \geq 128$
HN3$[\mathsf{SE1}, \mathsf{F}]$	Yes	No
HN4$[\mathsf{SE1}, \ell, \mathsf{F}]$		Yes
HN5$[\mathsf{TE}, \ell, \ell_z]$		Yes

Basic HN transforms.

We prove that all the following transforms turn a basic-AE1-secure NBE1 scheme SE1 into a basic-AE2-secure NBE2 scheme SE2. (Recall basic means nonces are unique, never reused across encryption queries.) Pseudocode and pictures for the transforms are in Figure 1.5.

Having first produced a core ciphertext C_1 under SE1, the idea of scheme $\mathsf{SE2} =$ **HN1**$[\mathsf{SE1}, \mathsf{F}]$ is to use C_1 itself as a nonce to encrypt the actual nonce in counter mode under PRF F. A drawback is that this requires the minimal core-ciphertext length SE1.mccl to be non-trivial, like at least 128, which is not true for all SE1. Scheme $\mathsf{SE2} =$ **HN2**$[\mathsf{SE1}, \ell, \mathsf{E}, \mathsf{Spl}]$ turns to the perhaps more obvious idea of enciphering the nonce with a PRF-secure blockcipher E. The difficulty is the typicality of 96-bit nonces and 128-bit blockciphers, under which naïve enciphering would add a 32-bit ciphertext overhead, which we resolve by ciphertext stealing, ℓ representing the number of stolen bits (32 in our example) and Spl an ability to choose how the splitting is done. Scheme $\mathsf{SE2} =$ **HN3**$[\mathsf{SE1}, \mathsf{F}]$ uses the result of PRF F on the actual nonce as a derived nonce under which to run SE1. This is similar to SIV [125, 108]; the difference is to achieve AE2 rather than AE1 and to apply the PRF only to the nonce (rather than nonce, message and header) to have constant computational overhead.

Advanced HN transforms.

Unique nonces are easier to mandate in theory than assure in practice, where nonces may repeat due to errors, system resets, or replication. In that case (returning here to NBE1), not only does basic AE1-security give no security guarantees, but also damaging attacks are possible for schemes including CCM and GCM [86, 131]. Rogaway and Shrimpton's misuse resistant NBE1, which we refer to as advanced-AE1-secure NBE1, minimizes the damage from reused nonces, retaining AE1-security as long as no nonce, message, header triple is re-encrypted [125]. This still being for the NBE1 syntax, however, the concerns with adversary-visible nonces compromising message and user privacy are unchanged. We seek the NBE2 analogue, correspondingly defining and achieving advanced-AE2-secure NBE2 to provide protection against reused nonces while also hiding them.

With our framework, the definition is easy, calling for no new games; the goal is simply $\mathrm{AE2}[\mathcal{A}^{\mathrm{ae2}}_{\underline{\mathrm{u}}\text{-}\underline{\mathrm{nmh}}}]$-security where $\mathcal{A}^{\mathrm{ae2}}_{\underline{\mathrm{u}}\text{-}\underline{\mathrm{nmh}}}$ is the class of unique-nonce, message, header adversaries, meaning ones that do not repeat a query to their ENC oracle. The presence of well-analyzed advanced-AE1-secure NBE1 schemes [125, 80, 75, 74, 39] again motivates transforms rather than from-scratch designs.

We start by revisiting our basic-security preserving transforms, asking whether they also preserve advanced security, meaning, if the starting NBE1 scheme is advanced-AE1-secure, is the transformed NBE2 scheme advanced-AE2-secure? We show that for **HN1**, the answer is YES. We then show that it is YES also for **HN2** as long as the amount ℓ of stolen ciphertext is large enough. (In practical terms, at least 128.) For **HN3**, the answer is NO.

That **HN1** and **HN2** have these properties is good, but we would like to do better. (Limitations of the above are that **HN1** puts a lower bound on $\mathsf{SE1.mccl}$ that is not always met, and setting $\ell = 128$ in **HN2** with typical 96-bit nonces will call for a 224-bit blockcipher.) We offer **HN4** and **HN5**, showing they provide advanced AE2-security. Pseudocode and pictures are in Figure 1.11.

Scheme $\mathsf{SE2} = \mathbf{HN4}[\mathsf{SE1}, \ell, \mathsf{F}]$ uses the result of PRF F on the actual nonce, message

and header as a derived nonce for SE1. The difference with SIV [125, 108] is that what is encrypted under SE1 includes the actual nonce in order to hide it. The computational overhead stays constant because SE1 need provide only privacy, which it can do in one pass. Scheme SE2 $=$ **HN5**$[\mathsf{TE}, \ell, \ell_z]$ is different, using the encode-then-encipher paradigm [29] to set the ciphertext to an enciphering, under an arbitrary-input-length, tweakable cipher TE, of the nonce, message and ℓ_t-bits of redundancy, with the header as tweak. Instantiating TE via the very fast AEZ tweakable block cipher [80] yields correspondingly fast, advanced-AE2-secure NBE2.

Dedicated transform for GCM.

While our generic transforms are already able, with low overhead, to immunize GCM [105, 61] —by this we mean turn this basic-AE1-secure NBE1 scheme into a basic-AE2-secure NBE2 scheme— we ask if a dedicated transform —one that exploits the structure of GCM— can do even better. The goal is not just even lower cost overhead, but minimization of software changes. We show that simply pre-pending a block of 0s, of length equal to the nonce length, to the message, and then GCM-encrypting, provides basic-AE2-security. This means no new key materiel needs to be added, and existing encryption software can be used in a blackbox way. Ciphertext overhead remains zero. Decryption software does however need a change.

The proceedings version of our paper [27] had claimed basic-AE2-security of our GCM variant assuming the blockcipher E was prp-cca secure (also called strong prp-security, this means the adversary is allowed both forward and backward queries) and the hash family H was AXU. In this version, we do better, reducing the assumption on E to just PRF security, and that on H to computational AXU. The proof of security is greatly simplified by establishing privacy and authenticity separately, which suffices courtesy of our general decomposition result (Theorem 1). Privacy is easily reduced (Theorem 11) to that of GCM itself, allowing us to conclude it via known results on the latter [105, 84, 31, 103, 82] and in particular to inherit the good bounds of [82]. The proof of Theorem 12, establishing authenticity, is more invasive and in our view the most non-trivial proof in this paper.

Related work.

In a 2013 mailing list message, Bernstein [32] argues that the security definitions for authenticated encryption fail to fully capture practical requirements, giving sequence privacy leakage via sequence-number nonces as an explicit example. AE2-secure NBE2 addresses these concerns. Bernstein also proposed a solution that can be seen as a specific instantiation of our **HN2** transformation.

As a technical step in achieving security against release of unverified plaintext (RUP), Ashur, Dunkelman and Luykx (ADL) [13] use a syntax identical to NBE2, and their techniques bear some similarities with ours that we discuss further in Section 1.8.

The CAESAR competition's call for authenticated encryption schemes describes a syntax where encryption receives, in place of a nonce, a public message number (PMN) and a secret message number (SMN), decryption taking only the former [44]. The formalization of Namprempre, Rogaway and Shrimpton (NRS) [109] dubs this "AE5." In this light, an NBE1 scheme is a AE5 scheme without a SMN and an NBE2 scheme is an AE5 scheme without a PMN.

Possible future work.

The concerns we have raised with regard to a gap between theory and usage, and privacy vulnerabilities created by adversary-visible nonces in the latter, arise fundamentally from the choice of *syntax* represented by NBE1, and as such hold also in other contexts where an NBE1-style syntax is used. This includes AE secure under release of unverified plaintext [8], KDM-secure AE [25, 34, 55], robust AE [64], online AE [65, 81], committing AE [72, 60], indifferentiable AE [17], subtle AE [19], leakage-resilient AE [18, 34] and MiniAE [107]. A direction for future work is to treat these with an NBE2-style syntax (decryption does not get the nonce) to provide nonce hiding.

While our transforms can be applied to promote the advanced-AE1-secure AES-GCM-SIV NBE1 scheme [74] to an advanced-AE2-secure NBE2 scheme, the bounds we get are inferior

to those of [39]. Bridging this gap to get advanced-AE2-secure NBE2 with security bounds like [39] is a direction for future work. Another is to prove better bounds for the authenticity of our AE2-secure version of GCM, in the vein of those for GCM [103, 82].

1.2 Preliminaries

Notation and terminology.

By ε we denote the empty string. By $|Z|$ we denote the length of a string Z. If Z is a string then $Z[i..j]$ is bits i through j of Z if $1 \le i \le j \le |Z|$, and otherwise is ε. By $x\|y$ we denote the concatenation of strings x, y. If x, y are equal-length strings then $x \oplus y$ denotes their bitwise xor. If i is an integer then $\langle i \rangle_n \in \{0,1\}^n$ denotes the representation of i mod 2^n as a string of (exactly) n bits. (For example, $\langle 3 \rangle_4 = 0011$.) If S is a finite set, then $|S|$ denotes it size. We say that a set S is *length-closed* if, for any $x \in S$ it is the case that $\{0,1\}^{|x|} \subseteq S$. (This will be a requirement for message, header and nonce spaces.)

If D, R are sets and $f : D \to R$ is a function then its image is $\mathrm{Im}(f) = \{ f(x) : x \in D \} \subseteq R$. By $\mathrm{FUNC}(D,R)$ we denote the set of all functions $f : D \to R$. If $|D| = |R|$ then by $\mathrm{BFUNC}(D,R)$ we denote the set of all bijections $f : D \to R$. Then $\mathrm{PERM}(D) = \mathrm{BFUNC}(D,D)$ is the set of all permutations $\pi : D \to D$.

If X is a finite set, we let $x \leftarrow\!\!\$\, X$ denote picking an element of X uniformly at random and assigning it to x. Algorithms may be randomized unless otherwise indicated. If A is an algorithm, we let $y \leftarrow A^{\mathrm{O}_1,\cdots}(x_1,\ldots;\omega)$ denote running A on inputs $x_1,\ldots$ and coins ω, with oracle access to $\mathrm{O}_1,\ldots$, and assigning the output to y. By $y \leftarrow\!\!\$\, A^{\mathrm{O}_1,\cdots}(x_1,\ldots)$ we denote picking ω at random and letting $y \leftarrow A^{\mathrm{O}_1,\cdots}(x_1,\ldots;\omega)$. We let $[A^{\mathrm{O}_1,\cdots}(x_1,\ldots)]$ denote the set of all possible outputs of A when run on inputs $x_1,\ldots$ and with oracle access to $\mathrm{O}_1,\ldots$. An adversary is an algorithm. Running time is worst case, which for an algorithm with access to oracles means across all possible replies from the oracles. We use $\perp$ (bot) as a special symbol to denote rejection, and it is assumed to not be in $\{0,1\}^*$.

Games.

We use the code-based game-playing framework of BR [30]. A game G (see Fig. 1.1 for an example) starts with an optional INIT procedure, followed by a non-negative number of additional procedures called oracles, and ends with a FIN procedure. Execution of adversary A with game G consists of running A with oracle access to the game procedures, with the restrictions that A's first call must be to INIT (if present), its last call must be to FIN, and it can call these procedures at most once. The output of the execution is the output of FIN. By $\Pr[\mathrm{G}(A)]$ we denote the probability that the execution of game G with adversary A results in this output being the boolean true.

Note that our adversaries have no output. The role of what in other treatments is the adversary output is, for us, played by the query to FIN.

Different games may have procedures (oracles) with the same names. If we need to disambiguate, we may write G.O to refer to oracle O of game G.

In games, integer variables, set variables boolean variables and string variables are assumed initialized, respectively, to 0, the empty set $\emptyset$, the boolean false and $\perp$.

Reductions.

Proofs give reductions that take a G_2-adversary A_2 and specify (construct) a G_1-adversary A_1 that runs A_2 as a subroutine, itself responding to oracle queries of A_2. Let $\textsc{Init}, \mathrm{O1}_1, \ldots, \mathrm{O1}_{n_1}, \textsc{Fin}$ denote the oracles of G_1 and $\textsc{Init}, \mathrm{O2}_1, \ldots, \mathrm{O2}_{n_2}, \textsc{Fin}$ the oracles of G_2. Then we may write pseudocode of the form

Adversary $A_1^{\textsc{Init},\mathrm{O1}_1,\ldots,\mathrm{O1}_{n_1},\textsc{Fin}}$

$\vdots$

$A_2^{\textsc{Init}^*,\mathrm{O2}_1^*,\ldots,\mathrm{O2}_{n_2}^*,\textsc{Fin}^*}$ / Run A_2 with specified subroutines as oracles

$\vdots$

procedure $\textsc{Init}^*$ / Subroutine simulating G_2.INIT

$\vdots$

procedure $\mathrm{O2}_1^*(\ldots)$ // Subroutine simulating $\mathrm{G}_2.\mathrm{O2}_1$

$\vdots$

Here $\textsc{Init}^*, \mathrm{O2}_1^*, \ldots, \mathrm{O2}_{n_2}^*, \textsc{Fin}^*$ are subroutines, given in the code of A_1, that are responsible for simulating the corresponding oracles for A_2 in G_2, and will invoke A_1's oracles to do so. We adopt the convention that if a simulation is trivial, meaning $\mathrm{O2}_i^*(x)$ returns $\mathrm{O1}_j(x)$, then, in the superscripts to A_2, we simply write $\mathrm{O1}_j$ in place of $\mathrm{O2}_i^*$, and do not give code for the simulated oracle.

Multi-user security.

There is growing recognition that security should be considered in the multi-user (mu) setting [21] rather than the traditional single-user (su) one. Our main definitions are in the mu setting. The games provide the adversary a NEW oracle, calling which results in a new user being initialized, with a fresh key. Other oracles are enhanced (relative to the su setting) to take an additional argument i indicating the user (key). We assume that adversaries do not make oracle queries to users (also called sessions) they have not initialized.

Function families.

A function family F specifies a deterministic evaluation algorithm $\mathsf{F.Ev} : \{0,1\}^{\mathsf{F.kl}} \times \mathsf{F.D} \to \{0,1\}^{\mathsf{F.ol}}$ that takes a key K and input x to return output $\mathsf{F.Ev}(K,x)$, where $\mathsf{F.kl}$ is the key length, $\mathsf{F.D}$ is the domain and $\mathsf{F.ol}$ is the output length. We say that F is invertible if there is an inversion algorithm $\mathsf{F.In} : \{0,1\}^{\mathsf{F.kl}} \times \{0,1\}^{\mathsf{F.ol}} \to \mathsf{F.D} \cup \{\perp\}$ such that for all $K \in \{0,1\}^{\mathsf{F.kl}}$ we have (1) $\mathsf{F.In}(K, \mathsf{F.Ev}(K,x)) = x$ for all $x \in \mathsf{F.D}$, and (2) $\mathsf{F.In}(K,y) = \perp$ for all $y \notin \mathrm{Im}(\mathsf{F.Ev}(K,\cdot))$. We say that F is a permutation family if it is invertible and $\mathsf{F.D} = \{0,1\}^{\mathsf{F.ol}}$. In that case, we also refer to F as a block cipher and to $\mathsf{F.ol}$ as the block length of F, which we may denote $\mathsf{F.bl}$.

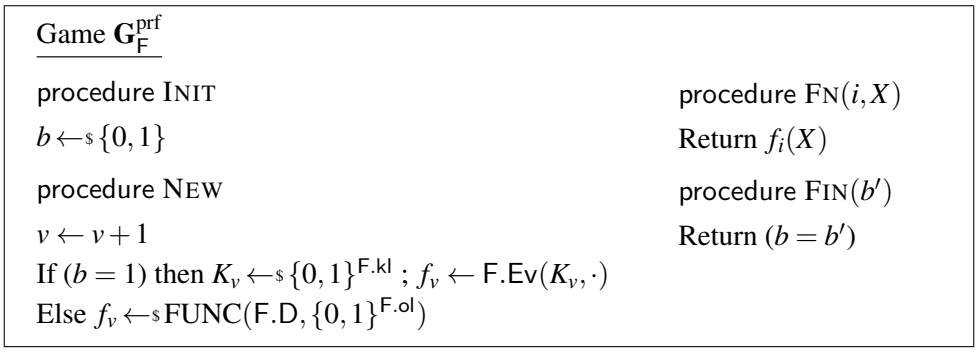

Figure 1.1. Game defining (multi-user) PRF security for function family F.

PRF security.

We define (multi-user) PRF security [23] for a function family F and adversary A via the game $\mathbf{G}^{\mathrm{prf}}_{\mathsf{F}}(A)$ in Fig. 1.1. Here b is the challenge bit. It is required that any FN(i,X) query of A satisfies $i \le v$ and $X \in \mathsf{F.D}$. The PRF advantage of adversary A is $\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(A) = 2\Pr[\mathbf{G}^{\mathrm{prf}}_{\mathsf{F}}(A)] - 1$.

1.3 Two frameworks for nonce-based encryption

We give definitions for both AE1-secure NBE1—current nonce-based encryption [124, 120, 122]— and AE2-secure NBE2—our new nonce-based encryption. In each case there is a single security game, different variant definitions then being captured by different adversary classes. This allows a unified and compact treatment.

NBE1.

An NBE1 scheme SE1 specifies several algorithms and related quantities, as follows. Deterministic encryption algorithm $\mathsf{SE1.Enc}: \mathsf{SE1.KS} \times \mathsf{SE1.NS} \times \mathsf{SE1.MS} \times \mathsf{SE1.HS} \to \{0,1\}^*$ takes a key K in the (finite) key-space SE1.KS, a nonce N in the nonce-space SE1.NS, a message M in the message space SE1.MS and a header H in the header space SE1.HS to return what we call a core ciphertext C_1. This is a string of length $\mathsf{SE1.ccl}(|N|,|M|,|H|)$, where SE1.ccl is the core-ciphertext length function. SE1 also specifies a deterministic decryption algorithm $\mathsf{SE1.Dec}: \mathsf{SE1.KS} \times \mathsf{SE1.NS} \times \{0,1\}^* \times \mathsf{SE1.HS} \to \mathsf{SE1.MS} \cup \{\perp\}$ that takes key K, nonce N,

core ciphertext C_1 and header H to return an output that is either a message $M \in \mathsf{SE1.MS}$, or $\perp$. It is required that $\mathsf{SE1.NS}, \mathsf{SE1.MS}, \mathsf{SE1.HS}$ are length-closed sets as defined in Section 1.2. Most often nonces are of a fixed length denoted $\mathsf{SE1.nl}$, meaning $\mathsf{SE1.NS} = \{0,1\}^{\mathsf{SE1.nl}}$. Decryption correctness requires that $\mathsf{SE1.Dec}(K, N, \mathsf{SE1.Enc}(K, N, M, H), H) = M$ for all $K \in \mathsf{SE1.KS}$, $N \in \mathsf{SE1.NS}$, $M \in \mathsf{SE1.MS}$ and $H \in \mathsf{SE1.HS}$.

AE1 game and advantage.

Let $\mathsf{SE1}$ be an NBE1 scheme and A an adversary. We associate to them the game $\mathbf{G}^{\mathrm{ae1}}_{\mathsf{SE1}}(A)$ shown on the top left of Fig. 1.2. (We use the name "AE1" to associate the game with the NBE1 syntax). The AE1-advantage of adversary A is $\mathbf{Adv}^{\mathrm{ae1}}_{\mathsf{SE1}}(A) = 2\Pr[\mathbf{G}^{\mathrm{ae1}}_{\mathsf{SE1}}(A)] - 1$. The game is in the multi-user setting, oracle NEW allowing the adversary to initialize a new user with a fresh key. It is required that any $\textsc{Enc}(i, N, M, H)$ query of A satisfy $1 \le i \le v$, $N \in \mathsf{SE1.NS}$, $M \in \mathsf{SE1.MS}$ and $H \in \mathsf{SE1.HS}$. When the challenge bit b is 1, the encryption oracle will return a core ciphertext as stipulated by $\mathsf{SE1.Enc}$, using the key for the indicated user i. In the $b = 0$ case, ENC will return a random string of length $\mathsf{SE1.ccl}(|N|, |M|, |H|)$. The array $\mathbf{M}$ is assumed to initially be $\perp$ everywhere, and holds core ciphertexts returned by ENC. It is required that any $\textsc{Dec}(i, N, C_1, H)$ query of A satisfy $1 \le i \le v$, $N \in \mathsf{SE1.NS}$ and $H \in \mathsf{SE1.HS}$. When the challenge bit b is 1, the decryption oracle will perform decryption as stipulated by $\mathsf{SE1.Dec}$, using the key for the indicated user i. In the $b = 0$ case, DEC will return $\perp$ on any core ciphertext not previously returned by the encryption oracle.

AE1 security metrics.

AE1-security is clearly not achievable without restrictions on the adversary. For example, if A repeats a query i, N, M, H to ENC, then, when $b = 1$ it gets back the same reply both times, while if $b = 0$ it likely does not, allowing it to determine b with high probability. We define different classes of adversaries, summarized by the table at the bottom of Figure 1.2, with the superscript "x" here being ae1. We say that NBE1 scheme $\mathsf{SE1}$ is AE1$[\mathcal{A}]$-secure if adversaries

Game $\mathbf{G}^{\text{ae1}}_{\mathsf{SE1}}$	Game $\mathbf{G}^{\text{ae2}}_{\mathsf{SE2}}$
procedure INIT	procedure INIT
$b \leftarrow\!\!\$\ \{0,1\}$	$b \leftarrow\!\!\$\ \{0,1\}$
procedure NEW	procedure NEW
$v \leftarrow v+1$; $K_v \leftarrow\!\!\$\ \mathsf{SE1.KS}$	$v \leftarrow v+1$; $K_v \leftarrow\!\!\$\ \mathsf{SE2.KS}$
procedure ENC(i,N,M,H)	procedure ENC(i,N,M,H)
If $(b=1)$ then	If $(b=1)$ then
$\quad C_1 \leftarrow \mathsf{SE1.Enc}(K_i,N,M,H)$	$\quad C_2 \leftarrow \mathsf{SE2.Enc}(K_i,N,M,H)$
$\quad$ Else $C_1 \leftarrow\!\!\$\ \{0,1\}^{\mathsf{SE1.ccl}(\lvert N\rvert,\lvert M\rvert,\lvert H\rvert)}$	$\quad$ Else $C_2 \leftarrow\!\!\$\ \mathsf{SE2.CS}(\lvert N\rvert,\lvert M\rvert,\lvert H\rvert)$
$\mathbf{M}[i,N,C_1,H] \leftarrow M$; Return C_1	$\mathbf{M}[i,C_2,H] \leftarrow M$; Return C_2
procedure DEC(i,N,C_1,H)	procedure DEC(i,C_2,H)
If $(\mathbf{M}[i,N,C_1,H] \neq \perp)$ then return $\mathbf{M}[i,N,C_1,H]$	If $(\mathbf{M}[i,C_2,H] \neq \perp)$ then return $\mathbf{M}[i,C_2,H]$
If $(b=0)$ then $M \leftarrow \perp$	If $(b=0)$ then $M \leftarrow \perp$
$\quad$ Else $M \leftarrow \mathsf{SE1.Dec}(K_i,N,C_1,H)$	$\quad$ Else $M \leftarrow \mathsf{SE2.Dec}(K_i,C_2,H)$
Return M	Return M
procedure FIN(b')	procedure FIN(b')
Return $(b=b')$	Return $(b=b')$

$\mathcal{A}^{\text{x}}_{\text{u-n}}$	Unique nonce adversaries — $A \in \mathcal{A}^{\text{x}}_{\text{u-n}}$ does not repeat a user-nonce pair i,N across its ENC queries
$\mathcal{A}^{\text{x}}_{\text{u-nmh}}$	Unique nonce-message-header adversaries — $A \in \mathcal{A}^{\text{x}}_{\text{u-nmh}}$ does not repeat a query to ENC
$\mathcal{A}^{\text{x}}_{\text{priv}}$	Privacy adversaries — $A \in \mathcal{A}^{\text{x}}_{\text{priv}}$ makes no DEC queries
$\mathcal{A}^{\text{x}}_{1}$	Single-user adversaries — $A \in \mathcal{A}^{\text{x}}_{1}$ makes only one NEW query
$\mathcal{A}^{\text{x}}_{\text{r-n}}$	Random-nonce adversaries — The nonces in the ENC queries of $A \in \mathcal{A}^{\text{x}}_{\text{r-n}}$ are distributed uniformly and independently at random

Figure 1.2. Game defining AE1-security of NBE1 scheme SE1, game defining AE2-security of NBE2 scheme SE2, and some classes of adversaries, leading to different security notions, where $\text{x} \in \{\text{ae1}, \text{ae2}\}$.

in $\mathcal{A}$ have low AE1-advantage. The definition is in the multi-user setting, but restricting attention to adversaries in the class $\mathcal{A}_1^{\mathrm{ae1}}$ allows us to recover the single-user setting. Different security notions in the literature are then captured as AE1$[\mathcal{A}]$-security for different classes of adversaries $\mathcal{A}$, as we illustrate below:

- $\mathcal{A}_{\mathrm{u\text{-}n}}^{\mathrm{ae1}}$ is the class of adversaries whose ENC queries never repeat a user-nonce pair. AE1$[\mathcal{A}_{\mathrm{u\text{-}n}}^{\mathrm{ae1}} \cap \mathcal{A}_1^{\mathrm{ae1}}]$-security is thus AEAD as defined in [120, 122].

- AE1$[\mathcal{A}_{\mathrm{u\text{-}n}}^{\mathrm{ae1}}]$-security is the extension of this to the multi-user setting as defined in [31], which we have referred to as basic AE1-security in Section 1.1.

- Adversaries in $\mathcal{A}_{\mathrm{u\text{-}nmh}}^{\mathrm{ae1}} \supseteq \mathcal{A}_{\mathrm{u\text{-}n}}^{\mathrm{ae1}}$ are allowed to re-use a user-nonce pair across ENC queries as long as they never repeat an entire query. AE1$[\mathcal{A}_{\mathrm{u\text{-}nmh}}^{\mathrm{ae1}} \cap \mathcal{A}_1^{\mathrm{ae1}}]$-security is misuse resistant AE [125].

- AE1$[\mathcal{A}_{\mathrm{u\text{-}nmh}}^{\mathrm{ae1}}]$-security is the extension of this to the multi-user setting [39], which we have referred to as advanced-AE1-security in Section 1.1.

- Adversaries in $\mathcal{A}_{\mathrm{r\text{-}n}}^{\mathrm{ae1}}$ pick the nonces in their ENC queries uniformly and independently at random from $\mathsf{SE1.NS}$. If $A \in \mathcal{A}_{\mathrm{r\text{-}n}}^{\mathrm{aeX}}$ then there is another adversary $\overline{A}$, called the core adversary, such that A runs as follows:

Adversary $A^{\mathrm{NEW},\mathrm{ENC},\mathrm{DEC}}$	procedure $\mathrm{ENC}^*(i,M,H)$
$\overline{A}^{\mathrm{NEW},\mathrm{ENC}^*,\mathrm{DEC}}$	$N \xleftarrow{\$} \mathsf{SE.NS}$; $C \leftarrow \mathrm{ENC}(i,N,M,H)$
	Return (N,C)

  No restriction is placed on how the adversary picks nonces in DEC queries. AE1$[\mathcal{A}_{\mathrm{r\text{-}n}}^{\mathrm{ae1}} \cap \mathcal{A}_1^{\mathrm{ae1}}]$-security is thus classical randomized AE [26] for schemes which make encryption randomness public, which is the norm.

- Sometimes, in the unique-nonce setting, we consider schemes that provide only privacy, not authenticity, and, rather than giving a separate game, can capture this as AE1$[\mathcal{A}_{\mathrm{priv}}^{\mathrm{ae1}} \cap \mathcal{A}_{\mathrm{u\text{-}n}}^{\mathrm{ae1}}]$-security. AE1$[\mathcal{A}_{\mathrm{priv}}^{\mathrm{ae1}} \cap \mathcal{A}_{\mathrm{u\text{-}n}}^{\mathrm{ae1}} \cap \mathcal{A}_1^{\mathrm{ae1}}]$-security is $\mathsf{IND\$\text{-}CPA}$ security, as defined in [120].

Further adversary classes can be defined to capture limited nonce reuse [39] or other resource restrictions.

We believe our (above) AE1 framework (single game, many adversary classes) is of independent interest, as a way to unify, better understand and compactly present existing and new notions of security for NBE1 schemes. We give a similar framework for AE2 next.

NBE2 syntax.

An NBE2 scheme $\mathsf{SE2}$ specifies several algorithms and related quantities, as follows. Deterministic encryption algorithm $\mathsf{SE2.Enc}: \mathsf{SE2.KS} \times \mathsf{SE2.NS} \times \mathsf{SE2.MS} \times \mathsf{SE2.HS} \to \{0,1\}^*$, just like for NBE1, takes a key K in the (finite) key-space $\mathsf{SE2.KS}$, a nonce N in the nonce-space $\mathsf{SE2.NS}$, a message M in the message space $\mathsf{SE2.MS}$ and a header H in the header space $\mathsf{SE2.HS}$ to return a ciphertext C_2 that is in the ciphertext space $\mathsf{SE2.CS}(|N|,|M|,|H|)$. $\mathsf{SE2}$ also specifies a deterministic decryption algorithm $\mathsf{SE2.Dec}: \mathsf{SE2.KS} \times \{0,1\}^* \times \mathsf{SE2.HS} \to \mathsf{SE2.MS} \cup \{\perp\}$ that takes key K, ciphertext C_2 and header H to return an output that is either a message $M \in \mathsf{SE2.MS}$, or $\perp$. (Unlike in NBE1, it does *not* take a nonce input.) It is required that $\mathsf{SE2.NS}, \mathsf{SE2.MS}, \mathsf{SE2.HS}$ are length-closed sets as defined in Section 1.2. Most often nonces are of a fixed length denoted $\mathsf{SE2.nl}$, meaning $\mathsf{SE2.NS} = \{0,1\}^{\mathsf{SE2.nl}}$. Decryption correctness requires that $\mathsf{SE2.Dec}(K, \mathsf{SE2.Enc}(K,N,M,H),H) = M$ for all $K \in \mathsf{SE2.KS}, N \in \mathsf{SE2.NS}, M \in \mathsf{SE2.MS}$ and $H \in \mathsf{SE2.HS}$.

AE2 game and advantage.

Let $\mathsf{SE2}$ be an NBE2 scheme and A an adversary. We associate to them the game $\mathbf{G}^{\mathrm{ae2}}_{\mathsf{SE2}}(A)$ shown on the top right of Fig. 1.2. (We use the name "AE2" to associate the game with the NBE2 syntax). The AE2-advantage of adversary A is $\mathbf{Adv}^{\mathrm{ae2}}_{\mathsf{SE2}}(A) = 2\Pr[\mathbf{G}^{\mathrm{ae2}}_{\mathsf{SE2}}(A)] - 1$. The game is in the multi-user setting, oracle NEW allowing the adversary to initialize a new user with a fresh key. It is required that any $\textsc{Enc}(i,N,M,H)$ query of A satisfy $1 \le i \le v$, $N \in \mathsf{SE2.NS}$, $M \in \mathsf{SE2.MS}$ and $H \in \mathsf{SE2.HS}$. When the challenge bit b is 1, the encryption oracle will return a

ciphertext as stipulated by SE2.Enc, using the key for the indicated user i. When $b = 0$, ENC will return a random element of the ciphertext space $\mathsf{SE2.CS}(|N|,|M|,|H|)$. The array $\mathbf{M}$ is assumed to initially be $\perp$ everywhere, and holds ciphertexts returned by ENC. It is required that any $\textsc{Dec}(i,C_2,H)$ query of A satisfy $1 \le i \le v$ and $H \in \mathsf{SE2.HS}$. When the challenge bit b is 1, the decryption oracle will perform decryption as stipulated by SE2.Dec, using the key for the indicated user i. When $b = 0$, DEC will return $\perp$ on any ciphertext not previously returned by the encryption oracle.

AE2 security metrics.

As with AE1-security, restrictions must be placed on the adversary to achieve AE2-security, and we use adversary classes to capture restrictions corresponding to different notions of interest. The classes are summarized by the table at the bottom of Figure 1.2, with the superscript "x" now being ae2. The classes and resulting notions are analogous to those for AE1. Thus, AE2$[\mathcal{A}^{\mathrm{ae2}}_{1}]$-security recovers the single-user setting. $\mathcal{A}^{\mathrm{ae2}}_{\text{u-n}}$ is the class of adversaries whose ENC queries never repeat a user-nonce pair, so AE2$[\mathcal{A}^{\mathrm{ae2}}_{\text{u-n}}]$-security is what we have referred to as basic AE2-security in Section 1.1. Adversaries in $\mathcal{A}^{\mathrm{ae2}}_{\text{u-nmh}} \supseteq \mathcal{A}^{\mathrm{ae2}}_{\text{u-n}}$ are allowed to re-use a user-nonce pair across ENC queries as long as they never repeat an entire query, so AE2$[\mathcal{A}^{\mathrm{ae2}}_{\text{u-nmh}}]$-security is what we have referred to as advanced AE2-security in Section 1.1. Adversaries in $\mathcal{A}^{\mathrm{ae2}}_{\text{r-n}}$ pick the nonces in their ENC queries uniformly and independently at random from SE2.NS. AE2$[\mathcal{A}^{\mathrm{ae2}}_{\text{priv}}]$-security is privacy only.

Discussion.

The main (small but important) change in the syntax from NBE1 to NBE2 is that in the latter, the decryption algorithm no longer gets the nonce as input. It is up to encryption to ensure that the ciphertext contains everything (beyond key and header) needed to decrypt. Nonces are thus no longer magically communicated, making the interface, and the task of application designers, simpler and less error-prone, reducing the possibility of loss of privacy

Game $\mathbf{G}^{\text{auth1}}_{\mathsf{SE1}}$	Game $\mathbf{G}^{\text{auth2}}_{\mathsf{SE2}}$
procedure NEW	procedure NEW
$v \leftarrow v+1$; $K_v \leftarrow\!\!\$\ \mathsf{SE1.KS}$	$v \leftarrow v+1$; $K_v \leftarrow\!\!\$\ \mathsf{SE2.KS}$
procedure ENC(i,N,M,H)	procedure ENC(i,N,M,H)
$C_1 \leftarrow \mathsf{SE1.Enc}(K_i,N,M,H)$	$C_2 \leftarrow \mathsf{SE2.Enc}(K_i,N,M,H)$
$\mathcal{S} \leftarrow \mathcal{S} \cup \{(i,N,C_1,H)\}$; Return C_1	$\mathcal{S} \leftarrow \mathcal{S} \cup \{(i,C_2,H)\}$; Return C_2
procedure VF(i,N,C_1,H)	procedure VF(i,C_2,H)
$M \leftarrow \mathsf{SE1.Dec}(K_i,N,C_1,H)$	$M \leftarrow \mathsf{SE2.Dec}(K_i,C_2,H)$
If $(M \neq \perp) \wedge ((i,N,C_1,H) \notin \mathcal{S})$ then	If $(M \neq \perp) \wedge ((i,C_2,H) \notin \mathcal{S})$ then
$\quad \mathsf{win} \leftarrow \mathsf{true}$	$\quad \mathsf{win} \leftarrow \mathsf{true}$
Return $(M = \perp)$	Return $(M = \perp)$
procedure FIN	procedure FIN
Return win	Return win

Figure 1.3. Games defining authenticity of NBE1 scheme SE1 (left) and NBE2 scheme SE2 (right).

from poor choices of nonces and opening the door to nonce-hiding security as captured by AE2. Another change is that, rather than a ciphertext length function, an NBE2 scheme specifies a ciphertext space. The reason is that a ciphertext might have some structure, like being a pair (C,C'). Ciphertexts like this cannot be indistinguishable from random strings, but they can be indistinguishable from pairs of random strings, which is captured by defining the ciphertext space correspondingly. This follows [72], in whose committing AE definition the same issue arose.

Nonce-Recovering NBE2.

A natural subclass of NBE2 schemes are those which recover the nonce explicitly during decryption. We provide definitions to capture such schemes. We say that an NBE2 scheme SE2 is nonce-recovering if there exists a deterministic nonce-plus-message recovery algorithm SE2.NMR such that for any $(K,C_2,H) \in \mathsf{SE2.KS} \times \{0,1\}^* \times \mathsf{SE2.HS}$, if $\mathsf{SE2.NMR}(K,C_2,H) \neq \perp$ then it parses as a pair $(M,N) \in \mathsf{SE2.MS} \times \mathsf{SE2.NS}$ satisfying $\mathsf{SE2.Dec}(K,C_2,H) = M$ and

$\mathsf{SE2.Enc}(K,N,M,H) = C_2$. Most of our transforms from NBE1 scheme to NBE2 schemes yield nonce-recovering NBE2 schemes.

1.4 Some general results

We give a few general results that we will use.

Priv+Auth implies AE.

Early definitions of authenticated encryption gave separate privacy and authenticity requirements [26]. Above, in the style of [120], a single game captures a joint privacy-and-authenticity requirement. Bose, Hoang and Tessaro (BHT) [39] showed that, for basic-secure AE1, separate, privacy-alone and authenticity alone conditions imply the joint condition. Here we extend this to both advanced security and AE2. This is useful because (1) It can be easier to establish the simpler, separate requirements than the joint one, and (2) Proven bounds can differ for privacy and authenticity, which is not visible if one only gives results for the joint notion.

Proceeding, the definition for privacy alone is already present, obtained above by restricting to adversaries in the classes $\mathrm{AE1}[\mathcal{A}^{\mathrm{ae1}}_{\mathrm{priv}}]$ (for NBE1) or $\mathrm{AE2}[\mathcal{A}^{\mathrm{ae2}}_{\mathrm{priv}}]$ (for NBE2). To define authenticity-alone, consider the games $\mathbf{G}^{\mathrm{auth1}}_{\mathsf{SE1}}$ and $\mathbf{G}^{\mathrm{auth2}}_{\mathsf{SE2}}$ in Fig. 1.3, where $\mathsf{SE1}$ is a NBE1 scheme and $\mathsf{SE2}$ is an NBE2 scheme. The auth1-advantage of adversary A is $\mathbf{Adv}^{\mathrm{auth1}}_{\mathsf{SE1}}(A) = \Pr[\mathbf{G}^{\mathrm{auth1}}_{\mathsf{SE1}}(A)]$. The auth2-advantage of adversary A is $\mathbf{Adv}^{\mathrm{auth2}}_{\mathsf{SE2}}(A) = \Pr[\mathbf{G}^{\mathrm{auth2}}_{\mathsf{SE2}}(A)]$.

As for AE, different notions of security are captured by considering different classes of adversaries. For $\mathrm{x} \in \{\mathrm{auth1}, \mathrm{auth2}\}$ we define:

- $\mathcal{A}^{\mathrm{x}}_{\text{u-n}}$ is the class of adversaries whose ENC queries never repeat a user-nonce pair.
- Adversaries in $\mathcal{A}^{\mathrm{x}}_{\text{u-nmh}} \supseteq \mathcal{A}^{\mathrm{x}}_{\text{u-n}}$ are allowed to re-use a user-nonce pair across ENC queries as long as they never repeat an entire query.
- Adversaries in $\mathcal{A}^{\mathrm{x}}_{\text{r-n}}$ pick the nonces in their ENC queries uniformly and independently at random from the nonce space of the scheme. Adversary $B \in \mathcal{A}^{\mathrm{authX}}_{\text{r-n}}$, analogous to

$A \in \mathcal{A}^{\text{aeX}}_{\text{r-n}}$ defines a core adversaries $\overline{B}$ and works as follows:

Adversary $B^{\text{NEW},\text{ENC},\text{VF}}$	procedure $\text{ENC}^*(i,M,H)$
$\overline{B}^{\text{NEW},\text{ENC}^*,\text{VF}}$	$N \xleftarrow{\$} \mathsf{SE.NS}$; $C \leftarrow \text{ENC}(i,N,M,H)$
	Return (N,C)

- $\mathcal{A}^{\text{x}}_{\text{ord}}$ is the class of adversaries that are *orderly*. An adversary is orderly if two conditions hold. First, it makes its VF queries after all its ENC queries. (That is, once it has made a VF query, it does not make any more ENC queries.) Second, the VF queries are non-adaptive, meaning a VF query does not depend on the answer to a prior VF query. (But the VF queries can depend on answers to the prior ENC queries). Intuitively, think of an orderly adversary as first making a bunch of ENC queries, and then a bunch of VF queries in parallel.

The following theorem says that AE-security decomposes into privacy plus authenticity. The statement covers AE1 and AE2 (via the choice of X) and basic and advanced (via the choice of y) security, so that the single statement encompasses four results.

BHT [39] give and prove this result for basic AE1 secure NBE1. Our bound is slightly better than theirs, dropping an added term, and we generalize to AE2 and advanced security. As with BHT [39], the theorem allows us to restrict attention to orderly authenticity adversaries, which later makes proving authenticity simpler.

Theorem 1 *Let* $\text{X} \in \{1,2\}$ *and* $\text{y} \in \{\text{n},\text{nmh}\}$. *Let* SE *be a NBEX scheme. Let* $A \in \mathcal{A}^{\text{aeX}}_{\text{u-y}}$ *be an adversary. Then, we can construct adversaries* $B \in \mathcal{A}^{\text{aeX}}_{\text{priv}} \cap \mathcal{A}^{\text{aeX}}_{\text{u-y}}$ *and* $C \in \mathcal{A}^{\text{authX}}_{\text{ord}} \cap \mathcal{A}^{\text{authX}}_{\text{u-y}}$ *such that:*

$$\mathbf{Adv}^{\text{aeX}}_{\mathsf{SE}}(A) \leq \mathbf{Adv}^{\text{aeX}}_{\mathsf{SE}}(B) + \mathbf{Adv}^{\text{authX}}_{\mathsf{SE}}(C) \ .$$

Adversary B preserves the resources of A. Adversary C is orderly. Additionally, it preserves query resources to NEW, ENC*, its queries to* VF *are those that A makes to* DEC*, and it preserves running time.*

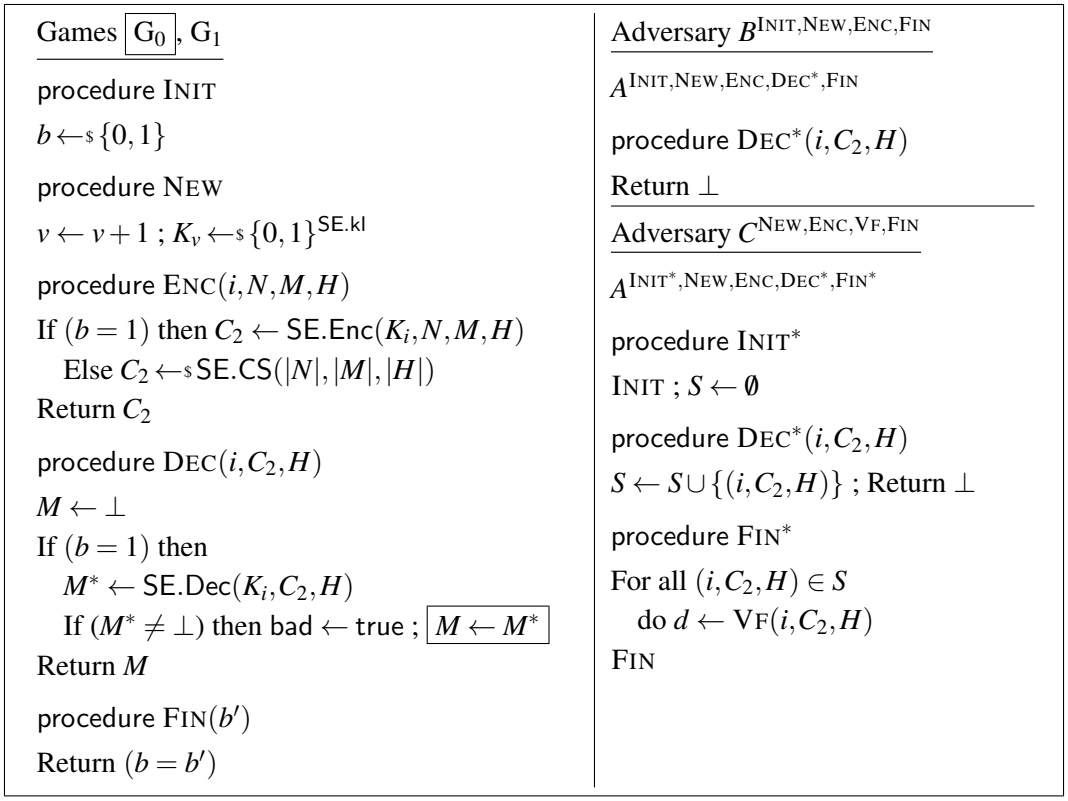

Figure 1.4. Games used in proving Theorem 1 (left) and Theorem 1 (right).

Proof. We give the proof for X=2, meaning for AE2. The proof for AE1 is analogous.

We assume that A makes no trivial queries. So it does not query $\textsc{Dec}(i, C_2, H)$ if $\mathbf{M}[i, C_2, H]$ is already defined. In the y=n case, it does not repeat a nonce-user pair in an $\textsc{Enc}$ query, and in the y=nmh case, it does not repeat an $\textsc{Enc}$ query. Games $\mathrm{G}_0, \mathrm{G}_1$ in Fig. 1.4 are identical-until-bad so using the Fundamental Lemma of Game Playing [30] we have

$$\begin{aligned}\mathbf{Adv}^{\mathrm{aeX}}_{\mathsf{SE}}(A) &= 2\Pr[\mathrm{G}_0(A)] - 1 \\ &= 2\Pr[\mathrm{G}_1(A)] - 1 + 2(\Pr[\mathrm{G}_0(A)] - \Pr[\mathrm{G}_1(A)]) \\ &\leq 2\Pr[\mathrm{G}_1(A)] - 1 + 2\Pr[\mathrm{G}_1(A) \text{ sets } \mathsf{bad}] \ .\end{aligned}$$

In Fig. 1.4, we specify adversary B such that

$$2\Pr[\mathrm{G}_1(A)] - 1 \leq \mathbf{Adv}^{\mathrm{aeX}}_{\mathsf{SE}}(B) \ .$$

Adversary B, being a privacy adversary, makes no $\textsc{Dec}$ queries, so we omit this oracle from the list in its superscript. It simulates all queries of A directly, except for additionally returning $\perp$ in response to any $\textsc{Dec}$ query made by A.

In game G_1, flag bad can only be set if $b = 1$, so

$$\begin{aligned}\Pr[\mathrm{G}_1(A) \text{ sets } \mathsf{bad}] &= \frac{1}{2} \cdot \Pr[\mathrm{G}_1(A) \text{ sets } \mathsf{bad} \,|\, b = 0] + \frac{1}{2} \cdot \Pr[\mathrm{G}_1(A) \text{ sets } \mathsf{bad} \,|\, b = 1] \\ &= \frac{1}{2} \cdot \Pr[\mathrm{G}_1(A) \text{ sets } \mathsf{bad} \,|\, b = 1] \ .\end{aligned}$$

In Fig. 1.4, we specify adversary C such that

$$\Pr[\mathrm{G}_1(A) \text{ sets } \mathsf{bad} \,|\, b = 1] \leq \mathbf{Adv}^{\mathrm{authX}}_{\mathsf{SE}}(C) \ .$$

Putting all this together concludes the proof. □

From single- to multi-user security.

The usual hybrid argument can be used to show that single-user security implies multi-user security up to a factor q_n degradation in advantage where q_n is the number of users, meaning the number of NEW queries of the adversary. As much as possible we will not rely on this but rather treat multi-user security directly, so as to avoid the degradation in the bound, but in some cases it will be easier to treat single-user security and take the hit in the bound. Accordingly we state the result here. We omit the proof since it is standard.

Theorem 2 *Let* $\mathrm{X} \in \{1,2\}$ *and* $\mathrm{y} \in \{\mathrm{n}, \mathrm{nmh}\}$. *Let* SE *be a NBEX scheme. Let* $A \in \mathcal{A}^{\mathrm{aeX}}_{\text{u-y}}$ *be an adversary making* q_n *calls to its* NEW *oracle and* q_e, q_d *calls per user to its* ENC *and* DEC *oracles, respectively. Then, we can construct adversary* $A \in \mathcal{A}^{\mathrm{aeX}}_{1} \cap \mathcal{A}^{\mathrm{aeX}}_{\text{u-y}}$ *such that:*

$$\mathbf{Adv}^{\mathrm{aeX}}_{\mathsf{SE}}(A) \leq q_n \cdot \mathbf{Adv}^{\mathrm{aeX}}_{\mathsf{SE}}(B) \ .$$

Adversary B makes 1 *query to its* NEW *oracle and* q_e, q_d *queries to its* ENC, DEC *oracles, respectively.*

Security under random nonces.

The following says that AE2$[\mathcal{A}^{\mathrm{ae2}}_{\text{u-n}}]$-security (resp. AE1$[\mathcal{A}^{\mathrm{ae1}}_{\text{u-n}}]$) implies AE2$[\mathcal{A}^{\mathrm{ae2}}_{\text{r-n}}]$-security (resp. AE1$[\mathcal{A}^{\mathrm{ae1}}_{\text{r-n}}]$) with a degradation in advantage corresponding to the probability that a nonce repeats for some user. We will refer to this later. We omit the (obvious) proof.

Theorem 3 *Let* $\mathrm{X} \in \{1,2\}$. *Let* SE *be a NBEX scheme. Let* $A_{\mathrm{rn}} \in \mathcal{A}^{\mathrm{aeX}}_{\text{r-n}}$ *be an adversary making* q_n *calls to its* NEW *oracle and* q_e *calls per user to its* ENC *oracle. Then, we can construct adversary* $A_{\mathrm{un}} \in \mathcal{A}^{\mathrm{aeX}}_{\text{u-n}}$ *such that*

$$\mathbf{Adv}^{\mathrm{aeX}}_{\mathsf{SE}}(A_{\mathrm{rn}}) \leq \mathbf{Adv}^{\mathrm{aeX}}_{\mathsf{SE}}(A_{\mathrm{un}}) + \frac{q_n q_e (q_e - 1)}{2^{\mathsf{SE.nl}}} \ .$$

Adversary A_{un} *preserves the resources of* A_{rn}.

Saying A_{un} preserves the resources of A_{rn} means that the number of queries to all oracles are the same for both, as is the running time.

1.5 Usage of NBE1: The Transmit-Nonce transform

With AE1-secure NBE1, the nonce is needed for decryption. But how does the decryptor get it? This is a question about usage not addressed in the formalism. The understanding, however, is that the nonce can be communicated in the clear, with the core ciphertext. One might argue this is fine because, in the AE1-formalism, the adversary picks the nonce, so seeing the nonce again in the ciphertext cannot give the adversary an advantage.

We have discussed in the introduction why this fails to model cases where the nonce is chosen by the user, and why, at least in general, nonce transmission may violate message privacy. But the claim, so far, was informal. The reason was that transmitting the nonce represents a *usage* of NBE1 and we had no definitions to capture this. With AE2-secure NBE2, that gap is filled and we are in a position to formalize the claim of usage insecurity.

Some readers may see this is unnecessary, belaboring an obvious point. Indeed, the intuition is clear enough. But formalizing it serves also as an introduction to exercising our framework. We capture the usage in question as an NBE2 scheme $\mathsf{SE}_{\mathrm{TN}} = \mathbf{TN}[\mathsf{SE1}]$ built from a given NBE1 scheme $\mathsf{SE1}$ by what we call the transmit-nonce transform **TN**. We detail the (rather obvious) claim that $\mathsf{SE}_{\mathrm{TN}}$ fails to meet AE2-security, and discuss how it will also fail to meet other, weaker privacy goals.

The TN transform.

Our **TN** (Transmit Nonce) transform takes an NBE1 scheme $\mathsf{SE1}$ and returns the NBE2 scheme $\mathsf{SE}_{\mathrm{TN}} = \mathbf{TN}[\mathsf{SE1}]$, that, as the name suggests, transmits the nonce in the clear, meaning the $\mathsf{SE}_{\mathrm{TN}}$ ciphertext is the nonce together with the $\mathsf{SE1}$ core ciphertext. In more detail, encryption algorithm $\mathsf{SE}_{\mathrm{TN}}.\mathsf{Enc}(K,N,M,H)$ lets $C_1 \leftarrow \mathsf{SE1}.\mathsf{Enc}(K,N,M,H)$ and returns ciphertext $C_2 \leftarrow (N,C_1)$. Decryption algorithm $\mathsf{SE}_{\mathrm{TN}}.\mathsf{Dec}(K,C_2,H)$ parses C_2 as a pair (N,C_1) with

$N \in \mathsf{SE1.NS}$ —we write this as $(N, C_1) \leftarrow C_2$— returning $\perp$ if the parsing fails, and else returning $M \leftarrow \mathsf{SE1.Dec}(K, N, C_1, H)$. NBE2 scheme $\mathsf{SE}_{\mathrm{TN}}$ has the same key space, message space and header space as SE1, and we define its ciphertext space via $\mathsf{SE}_{\mathrm{TN}}.\mathsf{CS}(\ell_n, \ell_m, \ell_h) = \mathsf{SE1.NS} \times \{0,1\}^{\mathsf{SE1.ccl}(\ell_n, \ell_m, \ell_h)}$ for all $\ell_n, \ell_m, \ell_h \geq 0$. Usage of SE1 in which the nonce is sent in the clear (along with the core ciphertext) can now be formally modeled by asking what formal security notions for NBE2 schemes are met by $\mathsf{SE}_{\mathrm{TN}} = \mathbf{TN}[\mathsf{SE1}]$.

Insecurity of TN[SE1].

Let SE1 be *any* NBE1 scheme. It might, like GCM, be AE1$[\mathcal{A}^{\mathrm{ae1}}_{\mathrm{u\text{-}n}}]$-secure, or it might even be AE1$[\mathcal{A}^{\mathrm{ae1}}_{\mathrm{u\text{-}nmh}}]$-secure. Regardless, we claim that NBE2 scheme $\mathsf{SE}_{\mathrm{TN}} = \mathbf{TN}[\mathsf{SE1}]$ fails to be AE2$[\mathcal{A}^{\mathrm{ae2}}_{\mathrm{priv}} \cap \mathcal{A}^{\mathrm{ae2}}_{\mathrm{u\text{-}n}}]$-secure, meaning fails to provide privacy even for adversaries that do not reuse a nonce. This is quite obvious, since the adversary can test whether the nonce in its ENC query matches the one returned in the ciphertext. In detail:

Adversary $A^{\mathrm{NEW},\mathrm{ENC}}$

INIT

Pick some $(N, M, H) \in \mathsf{SE1.NS} \times \mathsf{SE1.MS} \times \mathsf{SE1.HS}$ with $|N| \geq 1$

NEW // Initialize one user

$(N^*, C_1) \xleftarrow{\$} \mathrm{ENC}(1, N, M, H)$ // Ciphertext returned is a pair

If $(N^* = N)$ then $b' \leftarrow 1$ else $b' \leftarrow 0$

FIN(b')

This adversary has advantage $\mathbf{Adv}^{\mathrm{ae2}}_{\mathsf{SE}_{\mathrm{TN}}}(A) \geq 1 - 1/2 = 1/2$, so represents a violation of AE2$[\mathcal{A}^{\mathrm{ae2}}_{\mathrm{priv}} \cap \mathcal{A}^{\mathrm{ae2}}_{\mathrm{u\text{-}n}}]$-security.

Discussion.

The attack above may be difficult to reconcile with SE1 being AE1$[\mathcal{A}^{\mathrm{ae1}}_{\mathrm{u\text{-}n}}]$-secure, the question being that, in the AE1 game, the adversary picks the nonce, and thus already knows it, so why should seeing it again in the ciphertext give the adversary extra information? The

answer is that in usage the adversary does not know the nonce *a priori* and seeing may provide additional information. This is not modeled in AE1 but is modeled in AE2. To be clear, the above violation of AE2 security does *not* contradict the assumed AE1-security of $\mathsf{SE1}$.

One might (correctly) argue that AE2 is a strong requirement so failing it does not represent a concerning violation of security, but it is clear that $\mathsf{SE}_{\mathrm{TN}}$ will fail to meet even much weaker notions of privacy for NBE2 schemes that one could formalize in natural ways, such as message recovery security or semantic security. (The nonce could be message dependent, in the extreme equal to the message.) One might also suggest that the losses of privacy occur for pathological choices of nonces, and nonce transmission is just fine if the nonce is a random number or counter, to which there are two responses. (1) The pitch and promise of AE1$[\mathcal{A}^{\mathrm{ae1}}_{\mathrm{u\text{-}n}}]$-secure NBE1 is that *any* (non-repeating) nonce is fine. For example RBBK [124] says

> The entity that encrypts chooses a new nonce for every message with the *only* restriction that no nonce is used twice

and RFC 5116 says

> Applications SHOULD use the nonce formation method defined in Section 3.2, and MAY use any other method that meets the uniqueness requirement.

It is important to know (both to prevent misuse and for our understanding) that in usage of NBE1, security requires more than just uniqueness of nonces; one must be concerned with how they are conveyed to the receiver. (2) A counter nonce can lead to loss of user privacy, for example revealing identity information, that is resolved by moving to AE2$[\mathcal{A}^{\mathrm{ae2}}_{\mathrm{u\text{-}n}}]$-secure NBE2, which is nonce hiding.

Privacy violations of the type discussed here, and captured by **TN**, occur only when the nonce is transmitted in the clear. They do not arise in TLS, where the nonce is not transmitted. (It is a counter that is held, and locally updated, by both sender and receiver.)

1.6 Basic transforms

We have explained that AE2-secure NBE2 offers valuable security and usability benefits over current encryption. So we now turn to achieving it. We follow the development path of NBE1, first, in this section, targeting basic AE2-security —no user reuses a nonce, which in our framework corresponds to adversaries in the class $\mathcal{A}^{\mathrm{ae2}}_{\mathrm{u\text{-}n}}$— and then, in Section 1.7, targeting advanced AE2-security —misuse resistance, where nonce-reuse is allowed, which in our framework corresponds to adversaries in the class $\mathcal{A}^{\mathrm{ae2}}_{\mathrm{u\text{-}nmh}}$.

Significant effort has gone into the design and analysis of basic-AE1-secure NBE1 schemes. We want to leverage rather than discard this. Accordingly, rather than from-scratch designs, we seek *transforms* of basic-AE1-secure NBE1 schemes into basic-AE2-secure NBE2 ones. This section gives three transforms that are simple and efficient and minimize quantitative security loss.

1.6.1 Preliminaries

We assume for simplicity that the NBE1 schemes provided as input to our transforms have nonces of a fixed length, meaning that $\mathsf{SE1.NS} = \{0,1\}^{\mathsf{SE1.nl}}$. This holds for most real-world AE1-secure NBE1 schemes. All our transforms can be adapted to allow variable-length nonces.

Core ciphertexts in practical NBE1 schemes tend to be no shorter than a certain minimal value, for example 96 bits for typical usage of GCM with AES [61]. We refer to this value as the minimal core-ciphertext length of the scheme SE1, formally defining $\mathsf{SE1.mccl} = \min_{N,M,H}\{\mathsf{SE1.ccl}(|N|,|M|,|H|)\}$ where the minimum is over all $(N,M,H) \in \mathsf{SE1.NS} \times \mathsf{SE1.MS} \times \mathsf{SE1.HS}$. This is relevant because some of our transforms need SE1.mccl to be non-trivial to provide security.

All transforms here use two keys, meaning the key for the constructed NBE2 scheme SE2 is a pair consisting of a key for a PRF and a key for SE1. An implementation can, starting from a single overlying key, derive these sub-keys and store them, so that neither key size nor

computational cost increase. This is well understood and is done as part of OCB, GCM and many other designs.

The ciphertext overhead is the bandwidth cost of the transform. We now discuss how to measure it. In the NBE2 scheme $\mathsf{SE2}$ constructed by any of our transforms from an NBE1 scheme $\mathsf{SE1}$, the ciphertext space is the set of strings of some length, $\mathsf{SE2.CS}(\ell_n, \ell_m, \ell_h) = \{0,1\}^{\mathsf{SE2.cl}(\ell_n,\ell_m,\ell_h)}$. Since NBE1 decryption gets the nonce for free while NBE2 decryption must, effectively, communicate it via the ciphertext, the "fair" definition of the ciphertext overhead of the transform is the maximum, over all possible choices of ℓ_n, ℓ_m, ℓ_h, of

$$\mathsf{SE2.cl}(\ell_n, \ell_m, \ell_h) - \mathsf{SE2.ccl}(\ell_n, \ell_m, \ell_h) - \mathsf{SE1.nl}\ .$$

Another way to put it is that the ciphertext overhead is how much longer ciphertexts are in $\mathsf{SE2}$ than in $\mathbf{TN}[\mathsf{SE1}]$. All our transforms have ciphertext overhead zero, meaning are optimal in terms of bandwidth usage.

1.6.2 The HN1 transform

The idea of our first transform is that a piece of the core ciphertext may be used as a nonce under which to encrypt the actual nonce. Let $\mathsf{SE1}$ be an NBE1 scheme and F a function family with $\mathsf{F.ol} = \mathsf{SE1.nl}$, so that outputs of $\mathsf{F.Ev}$ can be used to mask nonces for $\mathsf{SE1}$. Assume $\mathsf{SE1.mccl} \geq \mathsf{F.il}$, so that an $\mathsf{F.il}$-bit prefix of a core ciphertext can be used as an input to $\mathsf{F.Ev}$. Invertibility of F is not required, so it can, but need not, be a blockcipher. Our **HN1** transform defines NBE2 scheme $\mathsf{SE}_{\mathrm{HN1}} = \mathbf{HN1}[\mathsf{SE1}, \mathsf{F}]$ whose encryption and decryption algorithms are shown in Figure 1.5. A key (K_{F}, K_1) for $\mathsf{SE}_{\mathrm{HN1}}$ is a pair consisting of a key K_{F} for F and a key K_1 for $\mathsf{SE1}$, so that the key space is $\mathsf{SE}_{\mathrm{HN1}}.\mathsf{KS} = \{0,1\}^{\mathsf{F.kl}} \times \mathsf{SE1.KS}$. The message, header and nonce spaces are unchanged. The parsing $Y \| C_1 \leftarrow C_2$ in the second line of the decryption algorithm $\mathsf{SE}_{\mathrm{HN1}}$ is such that $|Y| = \mathsf{SE1.nl}$. The ciphertext overhead is zero. The computational overhead is one call to $\mathsf{F.Ev}$ for each of encryption or decryption.

$\underline{\mathsf{SE}_{\mathsf{HN1}}.\mathsf{Enc}((K_{\mathsf{F}},K_1),N,M,H)}$	$\underline{\mathsf{SE}_{\mathsf{HN1}}.\mathsf{Dec}((K_{\mathsf{F}},K_1),C_2,H)}$
$C_1 \leftarrow \mathsf{SE1.Enc}(K_1,N,M,H)$	If $(\lvert C_2\rvert < \mathsf{SE1.nl}+\mathsf{F.il})$ then return $\perp$
$x \leftarrow C_1[1..\mathsf{F.il}]$; $P \leftarrow \mathsf{F.Ev}(K_{\mathsf{F}},x)$	$Y\|C_1 \leftarrow C_2$; $x \leftarrow C_1[1..\mathsf{F.il}]$; $P \leftarrow \mathsf{F.Ev}(K_{\mathsf{F}},x)$
$Y \leftarrow P \oplus N$; $C_2 \leftarrow Y\|C_1$	$N \leftarrow P \oplus Y$; $M \leftarrow \mathsf{SE1.Dec}(K_1,N,C_1,H)$
Return C_2	Return M
$\underline{\mathsf{SE}_{\mathsf{HN2}}.\mathsf{Enc}((K_{\mathsf{E}},K_1),N,M,H)}$	$\underline{\mathsf{SE}_{\mathsf{HN2}}.\mathsf{Dec}((K_{\mathsf{E}},K_1),C_2,H)}$
$C_1 \leftarrow \mathsf{SE1.Enc}(K_1,N,M,H)$	If $(\lvert C_2\rvert < \mathsf{E.bl})$ then return $\perp$
$(x,y) \leftarrow \mathsf{Spl.Ev}(\ell,C_1)$	$N\|x \leftarrow \mathsf{E.In}(K_{\mathsf{E}},C_2[1..\mathsf{E.bl}])$
$C_{2,1} \leftarrow \mathsf{E.Ev}(K_{\mathsf{E}},N\|x)$	$y \leftarrow C_2[(\mathsf{E.bl}+1)..\lvert C_2\rvert]$; $C_1 \leftarrow \mathsf{Spl.In}(x,y)$
$C_2 \leftarrow C_{2,1}\|y$; Return C_2	$M \leftarrow \mathsf{SE1.Dec}(K_1,N,C_1,H)$; Return M
$\underline{\mathsf{SE}_{\mathsf{HN3}}.\mathsf{Enc}((K_{\mathsf{F}},K_1),N,M,H)}$	$\underline{\mathsf{SE}_{\mathsf{HN3}}.\mathsf{Dec}((K_{\mathsf{F}},K_1),C_2,H)}$
$N_1 \leftarrow \mathsf{F.Ev}(K_{\mathsf{F}},N)$	If $(\lvert C_2\rvert < \mathsf{F.ol})$ then return $\perp$
$C_1 \leftarrow \mathsf{SE1.Enc}(K_1,N_1,M,H)$	$N_1\|C_1 \leftarrow C_2$; $M \leftarrow \mathsf{SE1.Dec}(K_1,N_1,C_1,H)$
$C_2 \leftarrow N_1\|C_1$; Return C_2	Return M

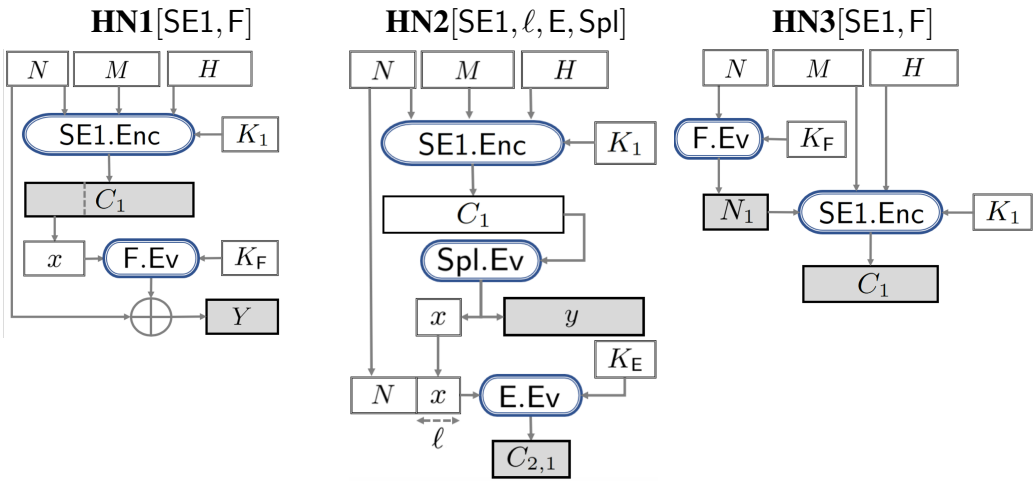

Figure 1.5. Pseudocode and pictorial descriptions of NBE2 schemes' algorithms. From top to bottom: $\mathsf{SE}_{\mathsf{HN1}} = \mathbf{HN1}[\mathsf{SE1},\mathsf{F}]$, $\mathsf{SE}_{\mathsf{HN2}} = \mathbf{HN2}[\mathsf{SE1},\ell,\mathsf{E},\mathsf{Spl}]$ and $\mathsf{SE}_{\mathsf{HN3}} = \mathbf{HN3}[\mathsf{SE1},\mathsf{F}]$. .

Adversary $A_1^{\text{NEW},\text{ENC},\text{VF},\text{FIN}}$

$A_2^{\text{NEW}^*,\text{ENC}^*,\text{VF}^*,\text{FIN}}$

procedure NEW^*

$v \leftarrow v+1$; $K_{\mathsf{F},v} \xleftarrow{\$} \{0,1\}^{\mathsf{F.kl}}$; NEW

procedure $\text{ENC}^*(i,N,M,H)$

$C_1 \leftarrow \text{ENC}(i,N,M,H)$; $x \leftarrow C_1[1..\mathsf{F.il}]$; $Y \leftarrow N \oplus \mathsf{F.Ev}(K_{\mathsf{F},i},x)$; $C_2 \leftarrow Y \| C_1$

Return C_2

procedure $\text{VF}^*(i,C_2,H)$

If $(|C_2| < \mathsf{SE1.nl} + \mathsf{F.il})$ then return $\perp$

$Y \| C_1 \leftarrow C_2$; $x \leftarrow C_1[1..\mathsf{F.il}]$; $N \leftarrow Y \oplus \mathsf{F.Ev}(K_{\mathsf{F},i},x)$; Return $\text{VF}(i,N,C_1,H)$

Figure 1.6. Adversary A_1 used in proving Equation (1.1).

Theorem 4 below says that if the starting NBE1 scheme SE1 is basic-AE1-secure and F is a PRF then the NBE2 scheme SE_{HN1} returned by the transform is basic-AE2-secure. We show authenticity and privacy separately —taking advantage of Theorem 1 to obtain joint security— not just for simplicity, but because the bounds and assumptions under which security can be established are different. Authenticity of SE_{HN1} reduces tightly to that of SE1 and does not require PRF-security of F, as indicated by Equation (1.1). PRF-security of F is only required for privacy, where there is also an added term in the bound, as indicated by Equation (1.2).

Theorem 4 *Let* $\mathsf{SE}_{\text{HN1}} = \mathbf{HN1}[\mathsf{SE1},\mathsf{F}]$ *be obtained as above. Then, given adversary* $A_2 \in \mathcal{A}^{\text{auth2}}_{\text{u-n}}$ *we construct adversary* $A_1 \in \mathcal{A}^{\text{auth1}}_{\text{u-n}}$ *such that*

$$\mathbf{Adv}^{\text{auth2}}_{\mathsf{SE}_{\text{HN1}}}(A_2) \;\le\; \mathbf{Adv}^{\text{auth1}}_{\mathsf{SE1}}(A_1)\;. \tag{1.1}$$

Adversary A_2 *preserves the resources of* A_1*. Also, given adversary* $A_2 \in \mathcal{A}^{\text{ae2}}_{\text{u-n}} \cap \mathcal{A}^{\text{ae2}}_{\text{priv}}$*, making* q_n *queries to its* NEW *oracle and* q_e *queries per user to its* ENC *oracle, we construct adversaries* $A_1 \in \mathcal{A}^{\text{ae1}}_{\text{u-n}} \cap \mathcal{A}^{\text{ae1}}_{\text{priv}}$ *and B such that*

$$\mathbf{Adv}^{\text{ae2}}_{\mathsf{SE}_{\text{HN1}}}(A_2) \;\le\; \mathbf{Adv}^{\text{ae1}}_{\mathsf{SE1}}(A_1) + \mathbf{Adv}^{\text{prf}}_{\mathsf{F}}(B) + \frac{q_n q_e(q_e-1)}{2^{\mathsf{F.il}+1}}\;. \tag{1.2}$$

procedure $\textsc{Fin}(b')$ // For all games
Return $(b' = 1)$

Games G_0, G_1

procedure $\textsc{New}$
$v \leftarrow v+1$; $K_{1,v} \leftarrow\!\!\$\ \mathsf{SE1.KS}$
$K_{\mathsf{F},v} \leftarrow\!\!\$\ \{0,1\}^{\mathsf{F.kl}}$; $f_v \leftarrow \mathsf{F.Ev}(K_{\mathsf{F},v},\cdot)$ // Game G_0
$f_v \leftarrow\!\!\$\ \mathrm{FUNC}(\{0,1\}^{\mathsf{F.il}},\{0,1\}^{\mathsf{F.ol}})$ // Game G_1

procedure $\textsc{Enc}(i,N,M,H)$
$C_1 \leftarrow \mathsf{SE1.Enc}(K_{1,i},N,M,H)$
$x \leftarrow C_1[1..\mathsf{F.il}]$; $P \leftarrow f_i(x)$; $Y \leftarrow P \oplus N$
$C_2 \leftarrow Y \| C_1$; Return C_2

Games G_2, $\boxed{G_3}$

procedure $\textsc{New}$
$v \leftarrow v+1$
$f_v \leftarrow\!\!\$\ \mathrm{FUNC}(\{0,1\}^{\mathsf{F.il}},\{0,1\}^{\mathsf{F.ol}})$

procedure $\textsc{Enc}(i,N,M,H)$
$C_1 \leftarrow\!\!\$\ \{0,1\}^{\mathsf{SE1.ccl}(|N|,|M|,|H|)}$
$x \leftarrow C_1[1..\mathsf{F.il}]$; $P \leftarrow f_i(x)$; $Y \leftarrow P \oplus N$
If $(x \in S_i)$ then bad $\leftarrow$ true ; $\boxed{Y \leftarrow\!\!\$\ \{0,1\}^{\mathsf{F.ol}}}$
$S_i \leftarrow S_i \cup \{x\}$
$C_2 \leftarrow Y \| C_1$; Return C_2

Adversary $B^{\textsc{Init},\textsc{New},\textsc{Fn},\textsc{Fin}}$

$A_2^{\textsc{Init},\textsc{New}^*,\textsc{Enc}^*,\textsc{Fin}}$

procedure $\textsc{New}^*$
$v \leftarrow v+1$; $K_{1,v} \leftarrow\!\!\$\ \mathsf{SE1.KS}$
$\textsc{New}$

procedure $\textsc{Enc}^*(i,N,M,H)$
$C_1 \leftarrow \mathsf{SE1.Enc}(K_{1,i},N,M,H)$
$x \leftarrow C_1[1..\mathsf{F.il}]$; $P \leftarrow \textsc{Fn}(i,x)$; $Y \leftarrow P \oplus N$
$C_2 \leftarrow Y \| C_1$; Return C_2

Adversary $A_1^{\textsc{Init},\textsc{New},\textsc{Enc},\textsc{Fin}}$

$A_2^{\textsc{Init},\textsc{New}^*,\textsc{Enc}^*,\textsc{Fin}}$

procedure $\textsc{New}^*$
$v \leftarrow v+1$
$f_v \leftarrow\!\!\$\ \mathrm{FUNC}(\{0,1\}^{\mathsf{F.il}},\{0,1\}^{\mathsf{F.ol}})$
$\textsc{New}$

procedure $\textsc{Enc}^*(i,N,M,H)$
$C_1 \leftarrow\!\!\$\ \textsc{Enc}(i,N,M,H)$
$x \leftarrow C_1[1..\mathsf{F.il}]$; $P \leftarrow f_i(x)$; $Y \leftarrow P \oplus N$
$C_2 \leftarrow Y \| C_1$; Return C_2

Figure 1.7. Games and adversaries used in proof of Equation (1.1). FIN is common to all games.

Adversary A_1 preserves the resources of A_2. Adversary B makes q_n queries to its NEW *oracle and q_e queries per user to its* FN *oracle. Adversary B has about the same running time as A_2.*

Proof. Adversary A_1 for the authenticity claim is in Figure 1.6. Adversary A_1's simulation of ENC queries is faithful. We need to check not only Equation (1.1) but also that A_1 belongs to $\mathcal{A}^{\text{auth1}}_{\text{u-n}}$. We claim that when a VF query of A_2 is winning (accepting and new) in its game, then the corresponding VF query of A_1 is winning (accepting and new) in its game. This comes down to the following. Fix K_{F} and C_1, let $x = C_1[1..\mathsf{F.il}]$ and let $Y, Y' \in \{0,1\}^{\mathsf{F.ol}}$. Let $N = Y \oplus \mathsf{F.Ev}(K_{\mathsf{F}}, x)$ and $N' = Y' \oplus \mathsf{F.Ev}(K_{\mathsf{F}}, x)$. Then $Y = Y'$ iff $N = N'$. Intuitively, with K_{F}, C_1 fixed, there is a one-to-one correspondence between full ciphertexts $Y \| C_1$ and nonce, core-ciphertext pairs (N, C_1) where $N = Y \oplus \mathsf{F.Ev}(K_{\mathsf{F}}, C_1[1..\mathsf{F.il}])$.

For the proof of privacy, consider the games in Fig. 1.7. Oracle DEC is dropped, since the privacy adversary makes no queries to it. Game G_0 is the real game. Game G_1 switches from F to random functions, which the adversary will not notice due to the assumed PRF security of F. Game G_2 switches to random core ciphertexts, which the adversary will not notice due to the assumed privacy of $\mathsf{SE1}$. Game G_3 switches to random full ciphertexts. Games $\mathrm{G}_2, \mathrm{G}_3$ differ only in the boxed code, so that the adversary notices the switch only when two calls to ENC pick the same value of x. This is exactly the probability that bad is set. Proceeding to the details, we have:

$$\begin{aligned}
&\mathbf{Adv}^{\text{ae2}}_{\mathsf{SE}_{\text{HN1}}}(A_2) = \Pr[\mathrm{G}_0(A_2)] - \Pr[\mathrm{G}_3(A_2)] \\
&= (\Pr[\mathrm{G}_0(A_2)] - \Pr[\mathrm{G}_1(A_2)]) \;+\; (\Pr[\mathrm{G}_1(A_2)] - \Pr[\mathrm{G}_2(A_2)]) \;+\; (\Pr[\mathrm{G}_2(A_2)] - \Pr[\mathrm{G}_3(A_2)]) \;.
\end{aligned}$$

Let adversaries A_1 and B be as in Fig. 1.7. For simplicity we show A_1 as picking f_v at random, but for efficiency (meaning, to keep the running time to the same as that of A_2) this must be

implemented via lazy sampling. Then:

$$\begin{aligned}
\Pr[\mathrm{G}_0(A_2)] - \Pr[\mathrm{G}_1(A_2)] &= \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(B)\ , \\
\Pr[\mathrm{G}_1(A_2)] - \Pr[\mathrm{G}_2(A_2)] &= \mathbf{Adv}^{\mathrm{ae1}}_{\mathsf{SE1}}(A_1)\ , \\
\Pr[\mathrm{G}_2(A_2)] - \Pr[\mathrm{G}_3(A_2)] &\leq \Pr[\mathrm{G}_2(A_2) \text{ sets } \mathsf{bad}] \\
&\leq \frac{q_n q_e(q_e-1)}{2^{\mathsf{F.il}+1}}\ .
\end{aligned}$$

The third inequality above used the Fundamental Lemma of Game Playing [30]. Putting the above together yields Equation (1.1). □

1.6.3 The HN2 transform

Splitting.

This transform employs ciphertext stealing [106] to get zero ciphertext overhead. There are many choices with regard to how to implement stealing, for example whether one steals from the first part of the core ciphertext or the last, and implementations may have different preferences. Accordingly, we do not pin down a choice but instead parameterize the transform by a splitting algorithm responsible for splitting a given string X (the core ciphertext) into segments x (the stolen part, of a prescribed length ℓ) and y (the rest). Formally, splitting scheme Spl specifies a deterministic algorithm $\mathsf{Spl.Ev}$ that takes an integer $\ell \geq 0$ and a string X with $|X| \geq \ell$, and returns a pair of strings $(x,y) \leftarrow \mathsf{Spl.Ev}(\ell, X)$ with $|x| = \ell$. If $(x,y) \in \mathrm{Im}(\mathsf{Spl.Ev}(|x|,\cdot))$ —the image of a function was defined in Section 1.2— then $X \leftarrow \mathsf{Spl.In}(x,y)$ recovers the unique X such that $\mathsf{Spl.Ev}(|x|,X) = (x,y)$, and otherwise returns $X = \perp$.

This isn't enough because for security we want that if X is random then so are x, y. A simple way to ensure this is to require that the split sets x to some bit positions of X and y to the rest, with the choice of positions depending only on $|X|$. Formally, we require that there is a (deterministic) function $\mathsf{Spl.St}$ that given integers ℓ, n with $n \geq \ell \geq 0$ returns a starting index

$s = \mathsf{Spl.St}(\ell, n)$ in the range $1 \le s \le n - \ell + 1$, and $\mathsf{Spl.Ev}(\ell, X)$ returns $x = X[s..(s+\ell-1)]$ and $y = X[1..(s-1)] \| X[(s+\ell)..|X|]$ for $s = \mathsf{Spl.St}(\ell, |X|)$. The most common choices are that $\mathsf{Spl.St}(\ell, n) = 1$, so that $x = X[1..\ell]$ is the ℓ-bit prefix of X and $y = X[(\ell+1)..|X|]$ is the rest (corresponding to stealing from the first part of X), or $\mathsf{Spl.St}(\ell, n) = n - \ell + 1$, so that $x = X[(|X| - \ell + 1)..|X|]$ is the ℓ-bit suffix of X and $y = X[1..(|X| - \ell)]$ is the rest (corresponding to stealing from the last part of X), but other choices are possible. Notice that now, assuming it is given inputs of the right lengths, as it will in our usage, $\mathsf{Spl.In}$ will not return $\perp$.

The HN2 transform.

The starting idea of this transform is that our NBE2 scheme can encrypt under the given NBE1 scheme and then also include in the ciphertext an enciphering, under a blockcipher E, of the nonce. We enhance this to encipher, along with the nonce, ℓ bits stolen from the core ciphertext. The stealing has two dividends. First, nonces are often shorter than the block length of E —for example $\mathsf{SE1.nl} = 96$ and $\mathsf{E.bl} = 128$ for AES-GCM and OCB [124, 99]— so in the absence of stealing, the nonce would be padded before enciphering, leading to ciphertext overhead. Second, while we show here (Theorem 5) that the scheme preserves basic security regardless of the amount ℓ stolen, we show later (Theorem 8) that it preserves even advanced security if ℓ is non-trivial (128 bits or more). We now proceed to the full description.

Let $\mathsf{SE1}$ be an NBE1 scheme, Spl a splitting scheme and $\ell \ge 0$ the prescribed length of the stolen segment of the core ciphertext. We assume the minimal core-ciphertext length of $\mathsf{SE1}$ satisfies $\mathsf{SE1.mccl} \ge \ell$, which ensures that core ciphertexts are long enough to allow the desired splitting. Let E be a blockcipher with block length $\mathsf{E.bl} = \mathsf{SE1.nl} + \ell$. Our **HN2** transform defines NBE2 scheme $\mathsf{SE}_{\mathrm{HN2}} = \mathbf{HN2}[\mathsf{SE1}, \ell, \mathsf{E}, \mathsf{Spl}]$ whose encryption and decryption algorithms are shown in Figure 1.5. The parsing in the second line of the decryption algorithm $\mathsf{SE}_{\mathrm{HN2}}$ is such that $|N| = \mathsf{SE1.nl}$. A key (K_{E}, K_1) for $\mathsf{SE}_{\mathrm{HN2}}$ is a pair consisting of a key K_{E} for E and a key K_1 for $\mathsf{SE1}$, so that the key space is $\mathsf{SE}_{\mathrm{HN2}}.\mathsf{KS} = \{0,1\}^{\mathsf{E.kl}} \times \mathsf{SE1.KS}$. The nonce, message and header spaces are unchanged. The length of ciphertext C_2 is $\mathsf{E.bl} + |C_1| - \ell = |C_1| + \mathsf{SE1.nl}$,

so the ciphertext space is $\mathsf{SE}_{\mathrm{HN2}}.\mathsf{CS}(\ell_n,\ell_m,\ell_h) = \{0,1\}^{\mathsf{SE1.nl}+\mathsf{SE1.ccl}(\ell_n,\ell_m,\ell_h)}$. The ciphertext overhead is zero. The computational overhead is an extra blockcipher call for encryption and a blockcipher inverse for decryption.

A typical instantiation for basic security is $\mathsf{E} = \mathsf{AES}$, so that $\mathsf{E.bl} = 128$. Nonces would have length $\mathsf{SE1.nl} = 96$. We then set $\ell = 32$ and $\mathsf{Spl.St}(\ell,n) = 1$ for all n. This means $\mathsf{SE1.mccl}$ must be at least 32, which is true for all real-world schemes we know. This reduction in the required value of $\mathsf{SE1.mccl}$ for security is a benefit that **HN2** offers over **HN1**. Recall the latter needs $\mathsf{F.il} \geq \mathsf{SE1.mccl}$, and hence by Theorem 4 needs $\mathsf{SE1.mccl} \geq 128$, for the same security that **HN2** can offer with $\mathsf{SE1.mccl} \geq 32$.

Theorem 5 below says that if the starting NBE1 scheme SE1 is basic-AE1-secure and E is a PRF, then the NBE2 scheme $\mathsf{SE}_{\mathrm{HN2}}$ returned by the transform is basic-AE2-secure. (This holds regardless of the value of ℓ.) We establish authenticity and privacy separately to showcase the difference in assumptions. Thus authenticity, as per Equation (1.3) does not require security of the blockcipher E and reduces tightly to the authenticity of SE1. For privacy, which relies on PRF security of E, Equation (1.4) shows that the reduction is tight, the added term of Equation (1.2) no longer present. This better bound is another benefit of **HN2** over **HN1**.

Theorem 5 *Let* $\mathsf{SE}_{\mathrm{HN2}} = \mathbf{HN2}[\mathsf{SE1},\ell,\mathsf{E},\mathsf{Spl}]$ *be obtained as above. Then, given adversary* $A_2 \in \mathcal{A}^{\mathrm{auth2}}_{\mathtt{u\text{-}n}}$ *we construct adversary* $A_1 \in \mathcal{A}^{\mathrm{auth1}}_{\mathtt{u\text{-}n}}$ *such that*

$$\mathbf{Adv}^{\mathrm{auth2}}_{\mathsf{SE}_{\mathrm{HN2}}}(A_2) \;\leq\; \mathbf{Adv}^{\mathrm{auth1}}_{\mathsf{SE1}}(A_1)\;. \tag{1.3}$$

Adversary A_2 *preserves the resources of* A_1*. Also, given adversary* $A_2 \in \mathcal{A}^{\mathrm{ae2}}_{\mathtt{u\text{-}n}} \cap \mathcal{A}^{\mathrm{ae2}}_{\mathrm{priv}}$*, making* q_n *queries to its* NEW *oracle and* q_e *queries per user to its* ENC *oracle, we construct adversaries* $A_1 \in \mathcal{A}^{\mathrm{ae1}}_{\mathtt{u\text{-}n}} \cap \mathcal{A}^{\mathrm{ae1}}_{\mathrm{priv}}$ *and B such that*

$$\mathbf{Adv}^{\mathrm{ae2}}_{\mathsf{SE}_{\mathrm{HN2}}}(A_2) \;\leq\; \mathbf{Adv}^{\mathrm{ae1}}_{\mathsf{SE1}}(A_1) + \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{E}}(B)\;. \tag{1.4}$$

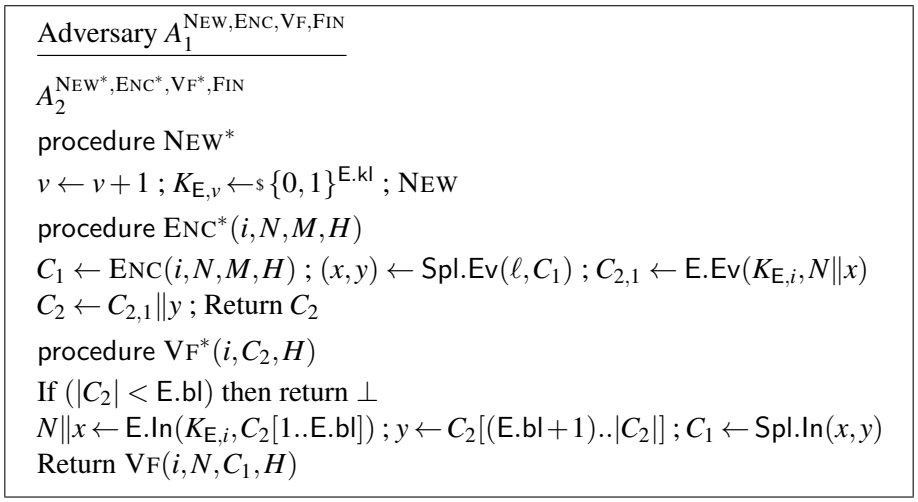

Figure 1.8. Adversary A_1 used in proving Equations (1.3) and (1.8).

Adversary A_1 preserves the resources of A_2. Adversary B makes q_n queries to its NEW *oracle and q_e queries per user to its* FN *oracle. Adversary B has about the same running time as A_2.*

Proof. Adversary A_1 for the authenticity claim of Equation (1.3) is in Figure 1.8.

For the proof of privacy, consider the games in Fig. 1.9. Oracle DEC is dropped, since the privacy adversary makes no queries to it. Game G_0 is the real game. Game G_1 switches from E to random functions, which the adversary will not notice due to the assumed PRF security of E. Game G_2 switches to random core ciphertexts, which the adversary will not notice due to the assumed privacy of $\mathsf{SE1}$. Game G_2 also has random full ciphertexts due to the uniqueness of nonces. Proceeding to the details, we have:

$$\begin{aligned}\mathbf{Adv}^{\text{ae2}}_{\mathsf{SE}_{\text{HN2}}}(A_2) &= \Pr[\mathrm{G}_0(A_2)] - \Pr[\mathrm{G}_2(A_2)] \\ &= (\Pr[\mathrm{G}_0(A_2)] - \Pr[\mathrm{G}_1(A_2)]) \;+\; (\Pr[\mathrm{G}_1(A_2)] - \Pr[\mathrm{G}_2(A_2)])\;.\end{aligned}$$

Let adversaries A_1 and B be as in Fig. 1.9. For simplicity we show A_1 as picking f_v at random, but for efficiency (meaning, to keep its running time the same as that of A_2) this must be implemented

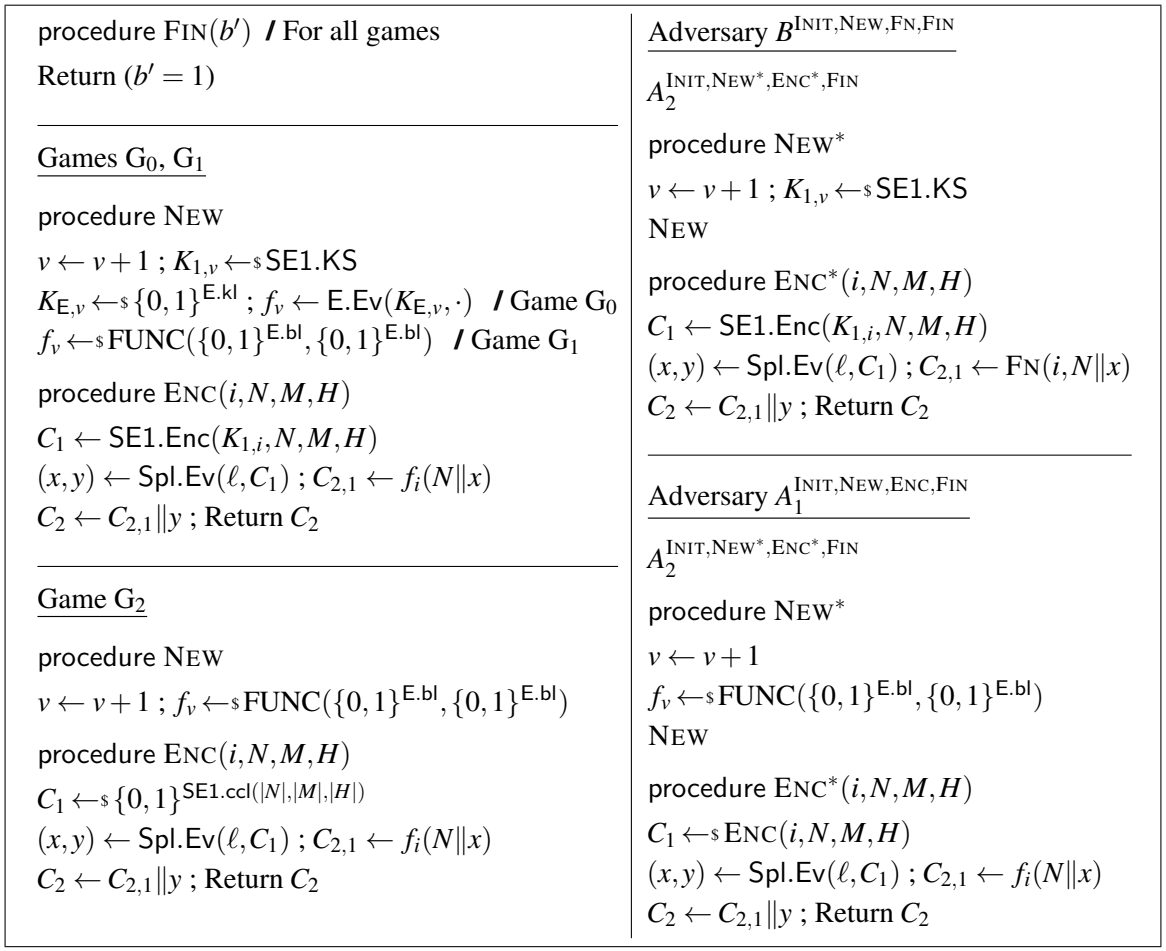

Figure 1.9. Games and adversaries used in proof of Equation (1.4). G_0, G_1 are also used in the proof of Equation (1.9). FIN are common to all games.

via lazy sampling. Then:

$$\Pr[\mathrm{G}_0(A_2)] - \Pr[\mathrm{G}_1(A_2)] = \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(B)\ ,$$
$$\Pr[\mathrm{G}_1(A_2)] - \Pr[\mathrm{G}_2(A_2)] = \mathbf{Adv}^{\mathrm{ae1}}_{\mathsf{SE1}}(A_1)\ .$$

Putting the above together yields Equation (1.4). □

1.6.4 The HN3 transform

Our third transform uses what we call nonce-based nonce-derivation, in which encryption is performed under $\mathsf{SE1}$ using as nonce the result $N_1 = \mathsf{F}(K_{\mathsf{F}}, N)$ of a PRF F on the actual nonce N. The idea comes from SIV [125] but differences include that: (1) SIV constructs an AE1-secure NBE1 scheme while we construct an AE2-secure NBE2 scheme. (2) SIV decryption needs to have the original nonce. (3) Our synthetic nonce N_1 is a function only of the actual nonce while the one in SIV is also a function of the message and header.

Proceeding to the details, let $\mathsf{SE1}$ be an NBE1 scheme. Let F be a function family with $\mathsf{F.ol} = \mathsf{SE1.nl}$, meaning outputs of $\mathsf{F.Ev}$ can be used as nonces for $\mathsf{SE1}$. Invertibility of F is not required, so it can, but need not, be a blockcipher. Our **HN3** transform defines NBE2 scheme $\mathsf{SE}_{\mathrm{HN3}} = \mathbf{HN3}[\mathsf{SE1}, \mathsf{F}]$ whose encryption and decryption algorithms are shown in Figure 1.5. A key (K_{F}, K_1) for $\mathsf{SE}_{\mathrm{HN3}}$ is a pair consisting of a key K_{F} for F and a key K_1 for $\mathsf{SE1}$, so that the key space is $\mathsf{SE}_{\mathrm{HN3}}.\mathsf{KS} = \{0,1\}^{\mathsf{F.kl}} \times \mathsf{SE1.KS}$. The message and header spaces are unchanged, and the nonce space is $\mathsf{SE}_{\mathrm{HN3}}.\mathsf{NS} = \{0,1\}^{\mathsf{F.il}}$, meaning inputs to F are nonces for $\mathsf{SE2}$. The parsing in the second line of the decryption algorithm $\mathsf{SE}_{\mathrm{HN3}}$ of Figure 1.5 is such that $|N_1| = \mathsf{SE1.nl}$. Note that the decryption algorithm does not use F or K_{F}.

As with **HN1** and **HN2**, the **HN3** transform has zero ciphertext overhead. The computational overhead for encryption is one invocation of F. Advantages emerge with decryption, where there is now *no* computational overhead. Indeed decryption in $\mathsf{SE}_{\mathrm{HN3}}$ is effectively the same as in $\mathsf{SE1}$. In particular, in the typical case that F is a blockcipher on which $\mathsf{SE1}$ is itself

based, decryption (unlike with **HN2**) no longer needs to implement its inverse, which can be a benefit in hardware and for reducing code size.

The assumed PRF security of F means that the nonce N_1 provided to $\mathsf{SE1.Enc}$ is effectively random. This makes it simple and natural, in proving security, to assume $\mathsf{SE1}$ is $\mathrm{AE1}[\mathcal{A}^{\mathrm{ae1}}_{\mathrm{r\text{-}n}}]$-secure (recall this is AE1-security for the class of adversaries that pick the nonce at random). Theorem 6 below accordingly says that if the starting NBE1 scheme $\mathsf{SE1}$ is $\mathrm{AE1}[\mathcal{A}^{\mathrm{ae1}}_{\mathrm{r\text{-}n}}]$-secure and F is a PRF then the NBE2 scheme $\mathsf{SE}_{\mathrm{HN1}}$ returned by the transform is basic-AE2-secure. The gap to the assumed basic-AE1-security of $\mathsf{SE1}$ is bridged by applying Theorem 3.

Theorem 6 *Let* $\mathsf{SE}_{\mathrm{HN3}} = \mathbf{HN3}[\mathsf{SE1}, \mathsf{F}]$ *be obtained as above. Then, given adversary* $A_2 \in \mathcal{A}^{\mathrm{ae2}}_{\mathrm{u\text{-}n}}$*, making* q_n *queries to its* NEW *oracle and* q_e *queries per user to its* ENC *oracle, we construct adversaries* $A_1 \in \mathcal{A}^{\mathrm{ae1}}_{\mathrm{r\text{-}n}}$ *and B such that*

$$\mathbf{Adv}^{\mathrm{ae2}}_{\mathsf{SE}_{\mathrm{HN3}}}(A_2) \;\le\; \mathbf{Adv}^{\mathrm{ae1}}_{\mathsf{SE1}}(A_1) + \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(B)\;. \tag{1.5}$$

Adversary A_1 *preserves the resources of* A_2*. Adversary B makes* q_n *queries to its* NEW *oracle and* q_e *queries per user to its* FN *oracle. Adversary B has about the same running time as* A_2*.*

Proof. We assume A_2 does not make trivial queries, meaning it does not make query $\mathrm{DEC}(i, C_2, H)$ if it has previously received C_2 in response to an $\mathrm{ENC}(i, \cdot, \cdot, H)$ query. Consider the games in Fig. 1.10. Game G_0 is the real game. Game G_1 switches from F to random functions, which the adversary will not notice due to the assumed PRF security of F. Game G_2 switches to random core ciphertexts and $\perp$ replies to DEC queries, which the adversary will not notice due to the assumed $\mathrm{AE1}[\mathcal{A}^{\mathrm{ae1}}_{\mathrm{r\text{-}n}}]$-security of $\mathsf{SE1}$. Game G_2 also has random full ciphertexts due to the uniqueness of nonces. Proceeding to the details, we have:

$$\begin{aligned}
&\mathbf{Adv}^{\mathrm{ae2}}_{\mathsf{SE}_{\mathrm{HN3}}}(A_2) = \Pr[\mathrm{G}_0(A_2)] - \Pr[\mathrm{G}_2(A_2)] \\
&= (\Pr[\mathrm{G}_0(A_2)] - \Pr[\mathrm{G}_1(A_2)]) \;+\; (\Pr[\mathrm{G}_1(A_2)] - \Pr[\mathrm{G}_2(A_2)])\;.
\end{aligned}$$

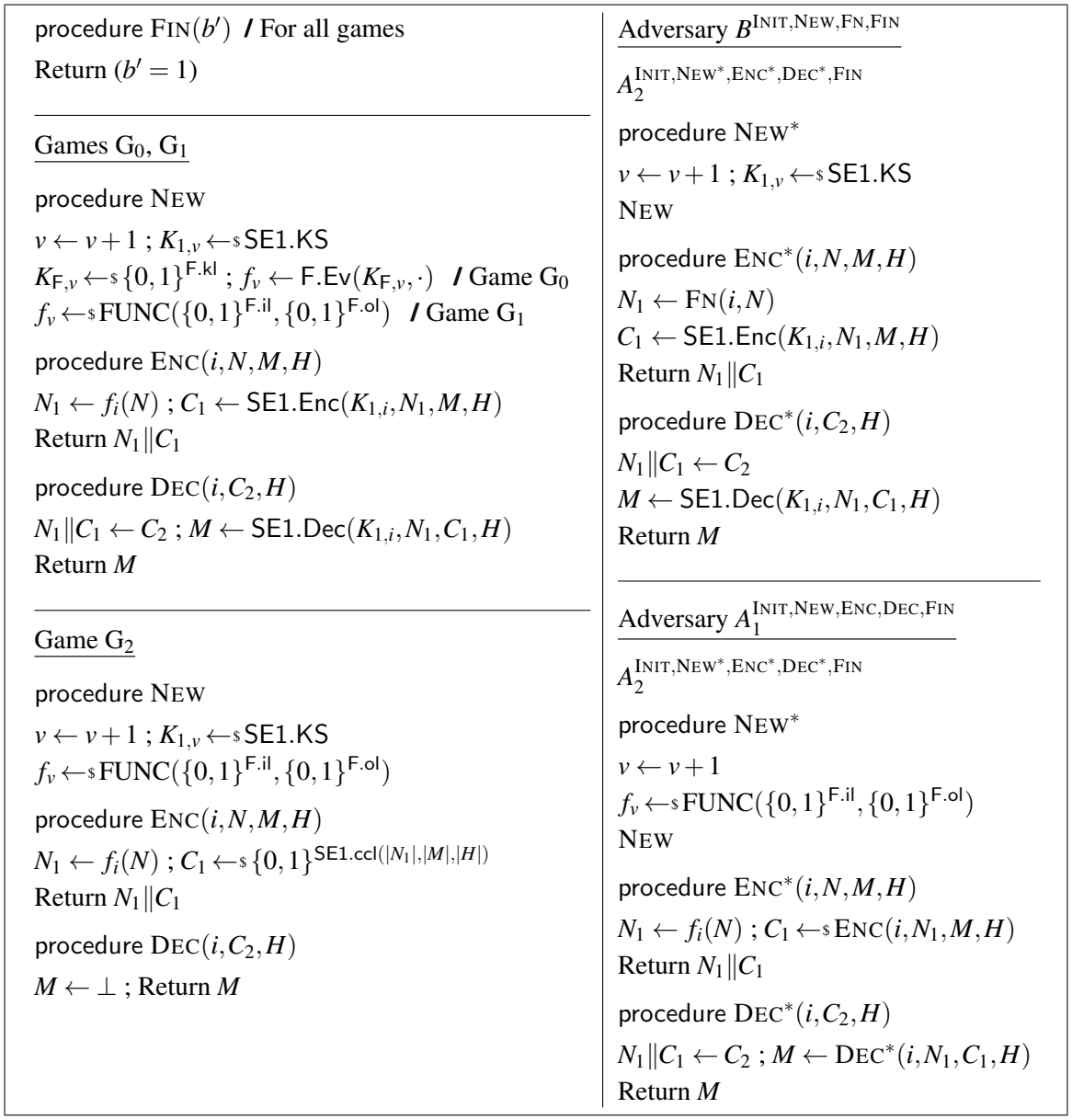

Figure 1.10. Games and adversaries used in proof of Equation (1.5). FIN is common to all games.

Let adversaries A_1 and B be as in Fig. 1.10. For simplicity we show A_1 as picking f_v at random, but for efficiency (meaning, to keep the running time to the same as that of A_2) this must be implemented via lazy sampling. Adversary A_1 is in the class $\mathcal{A}^{\text{ae1}}_{\text{r-n}}$ because the nonces it uses in its ENC queries are results of f_i on unique nonces, and are hence random and independent. Then:

$$\Pr[\mathrm{G}_0(A_2)] - \Pr[\mathrm{G}_1(A_2)] = \mathbf{Adv}^{\text{prf}}_{\mathsf{F}}(B) \ ,$$
$$\Pr[\mathrm{G}_1(A_2)] - \Pr[\mathrm{G}_2(A_2)] = \mathbf{Adv}^{\text{ae1}}_{\mathsf{SE1}}(A_1) \ .$$

Putting the above together yields Equation (1.5). □

1.7 Advanced transforms

We now turn to achieving AE2-security in the nonce-misuse setting, which we formalized as AE2$[\mathcal{A}^{\text{ae2}}_{\text{u-nmh}}]$-security. We discuss various transforms for this purpose.

1.7.1 Advanced security of HN1

We showed in Theorem 4 that **HN1** preserves basic security. It turns out that it also preserves advanced security. Theorem 7 below says that if the starting NBE1 scheme $\mathsf{SE1}$ is advanced-AE1-secure and F is a PRF then the NBE2 scheme SE_{HN1} returned by the transform is advanced-AE2-secure. The change in the statement compared to Theorem 4 is only with regard to the adversary classes changing from unique nonce (basic security) to unique nonce-message-header (advanced security). Again, Equation (1.6) tightly reduces authenticity of SE_{HN1} to that of $\mathsf{SE1}$ and makes no security assumptions on F, while the privacy claim of Equation (1.7) relies on PRF-security of F. The proof is very similar to that of Theorem 4 so we omit it for brevity.

Theorem 7 *Let* $\mathsf{SE}_{\text{HN1}} = \mathbf{HN1}[\mathsf{SE1}, \mathsf{F}]$ *be obtained as above. Then, given adversary* $A_2 \in \mathcal{A}^{\text{auth2}}_{\text{u-nmh}}$ *we construct adversary* $A_1 \in \mathcal{A}^{\text{auth1}}_{\text{u-nmh}}$ *such that*

$$\mathbf{Adv}^{\text{auth2}}_{\mathsf{SE}_{\text{HN1}}}(A_2) \ \leq \ \mathbf{Adv}^{\text{auth1}}_{\mathsf{SE1}}(A_1) \ . \tag{1.6}$$

$\underline{\mathsf{SE}_{\mathrm{HN4}}.\mathsf{Enc}((K_{\mathsf{F}},K_1),N,M,H)}$	$\underline{\mathsf{SE}_{\mathrm{HN4}}.\mathsf{Dec}((K_{\mathsf{F}},K_1),C_2,H)}$
$N_1 \leftarrow \mathsf{F.Ev}(K_{\mathsf{F}},(N,M,H))$	If $(\lvert C_2\rvert < \mathsf{F.ol})$ then return $\perp$
$C_1 \leftarrow \mathsf{SE1.Enc}(K_1,N_1,N\|M,H)$	$N_1\|C_1 \leftarrow C_2$; $X \leftarrow \mathsf{SE1.Dec}(K_1,N_1,C_1,H)$
$C_2 \leftarrow N_1\|C_1$	If $(X=\perp)$ then return $\perp$
Return C_2	$N\|M \leftarrow X$; $T \leftarrow \mathsf{F.Ev}(K_{\mathsf{F}},(N,M,H))$
	If $(T=N_1)$ then return M else return $\perp$
$\underline{\mathsf{SE}_{\mathrm{HN5}}.\mathsf{Enc}(K_{\mathsf{TE}},N,M,H)}$	$\underline{\mathsf{SE}_{\mathrm{HN5}}.\mathsf{Dec}(K_{\mathsf{TE}},C_2,H)}$
$C_2 \leftarrow \mathsf{TE.Ev}(K_{\mathsf{TE}},H,0^{\ell_z}\|N\|M)$	$X \leftarrow \mathsf{TE.In}(K_{\mathsf{TE}},H,C_2)$; If $X[1..\ell_z] \neq 0^{\ell_z}$ then return $\perp$
Return C_2	$N\|M \leftarrow X[(\ell_z+1)..\lvert X\rvert]$; Return M

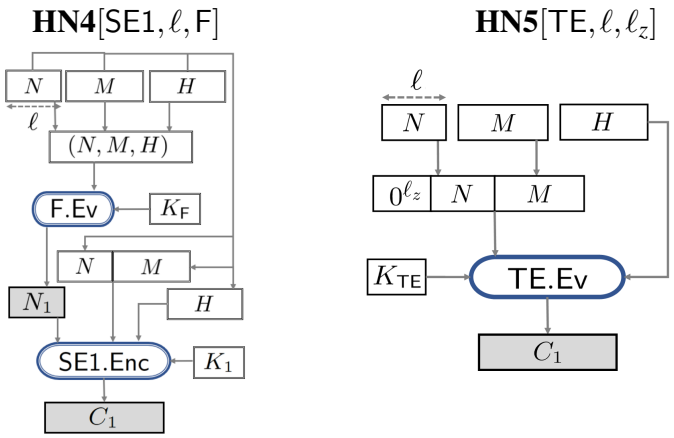

Figure 1.11. Pseudocode and pictorial descriptions of NBE2 schemes constructed using our advanced transforms. From top to bottom: $\mathsf{SE}_{\mathrm{HN4}} = \mathbf{HN4}[\mathsf{SE1},\ell,\mathsf{F}]$ and $\mathsf{SE}_{\mathrm{HN5}} = \mathbf{HN5}[\mathsf{TE},\ell,\ell_z]$.

Adversary A_2 preserves the resources of A_1. Also, given adversary $A_2 \in \mathcal{A}^{\text{ae2}}_{\text{u-nmh}} \cap \mathcal{A}^{\text{ae2}}_{\text{priv}}$, making q_n queries to its NEW *oracle and q_e queries per user to its* ENC *oracle, we construct adversaries $A_1 \in \mathcal{A}^{\text{ae1}}_{\text{u-nmh}} \cap \mathcal{A}^{\text{ae1}}_{\text{priv}}$ and B such that*

$$\mathbf{Adv}^{\text{ae2}}_{\mathsf{SE}_{\text{HN1}}}(A_2) \;\le\; \mathbf{Adv}^{\text{ae1}}_{\mathsf{SE1}}(A_1) + \mathbf{Adv}^{\text{prf}}_{\mathsf{F}}(B) + \frac{q_n q_e(q_e-1)}{2^{\mathsf{F.il}+1}} \;. \tag{1.7}$$

Adversary A_1 preserves the resources of A_2. Adversary B makes q_n queries to its NEW *oracle and q_e queries per user to its* FN *oracle. Adversary B has about the same running time as A_2.*

1.7.2 Advanced security of HN2

We showed in Theorem 5 that **HN2** preserves basic security regardless of the amount ℓ of stolen core-ciphertext, even $\ell = 0$. For small ℓ, however, **HN2** can leak information about the nonce in the advanced (misuse resistance) setting, so that the resulting scheme does not provide AE2$[\mathcal{A}^{\text{ae2}}_{\text{u-nmh}}]$-security.

To see how **HN2** can reveal information about the nonce, consider the case that $\ell = 0$. Now if two different message-header pairs are encrypted with the same nonce, then the first part of the ciphertext is the same, leading to an $\mathcal{A}^{\text{ae2}}_{\text{u-nmh}}$-adversary with advantage $1 - 2^{-\mathsf{E.bl}}$. The advantage of this attack however decreases (exponentially) as ℓ increases. The following theorem says that once ℓ is non-trivial (say, 128 bits or more), the transform actually preserves advanced security as well. The proof is very similar to that of Theorem 5 so we omit it for brevity.

Theorem 8 *Let $\mathsf{SE}_{\text{HN2}} = \mathbf{HN2}[\mathsf{SE1}, \ell, \mathsf{E}, \mathsf{Spl}]$ be obtained as above. Then, given adversary $A_2 \in \mathcal{A}^{\text{auth2}}_{\text{u-nmh}}$ 1 we construct adversary $A_1 \in \mathcal{A}^{\text{auth1}}_{\text{u-nmh}}$ such that*

$$\mathbf{Adv}^{\text{auth2}}_{\mathsf{SE}_{\text{HN2}}}(A_2) \;\le\; \mathbf{Adv}^{\text{auth1}}_{\mathsf{SE1}}(A_1) \;. \tag{1.8}$$

Adversary A_2 preserves the resources of A_1. Also, given adversary $A_2 \in \mathcal{A}^{\text{ae2}}_{\text{u-nmh}} \cap \mathcal{A}^{\text{ae2}}_{\text{priv}}$, making q_n queries to its NEW *oracle and q_e queries per user to its* ENC *oracle, we construct adversaries*

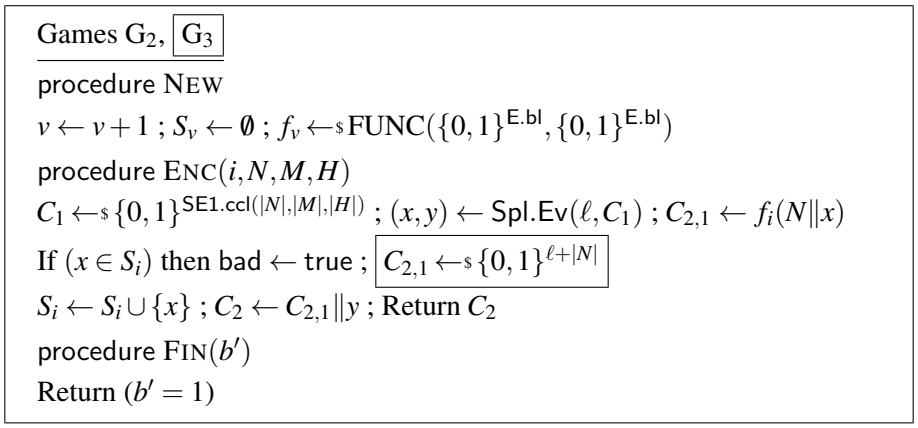

Figure 1.12. Games G_2, G_3 used in proof of Equation (1.9).

$A_1 \in \mathcal{A}^{\mathrm{ae1}}_{\text{u-nmh}} \cap \mathcal{A}^{\mathrm{ae1}}_{\mathrm{priv}}$ *and B such that*

$$\mathbf{Adv}^{\mathrm{ae2}}_{\mathsf{SE}_{\mathrm{HN2}}}(A_2) \;\le\; \mathbf{Adv}^{\mathrm{ae1}}_{\mathsf{SE1}}(A_1) + \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{E}}(B) + \frac{q_n q_e (q_e - 1)}{2^{\ell+1}} \,. \tag{1.9}$$

Adversary A_1 preserves the resources of A_2. Adversary B makes q_n queries to its NEW *oracle and q_e queries per user to its* FN *oracle. Adversary B has about the same running time as A_2.*

Proof. The proof of Theorem 8 is very similar to that of Theorem 5.

The adversary A_1 used in proving Equation (1.8) is the same one depicted in Fig. 1.8. Note that if $A_2 \in \mathcal{A}^{\mathrm{auth2}}_{\text{u-nmh}}$ then $A_1 \in \mathcal{A}^{\mathrm{auth1}}_{\text{u-nmh}}$, meaning that A_1 is in the desired adversary class.

For the proof of privacy, we will make use of games G_0, G_1 from the proof of Theorem 8 (Fig. 1.9), but define the new games G_2, G_3 shown in Fig. 1.12). As before, G_0 is the real game, while game G_1 switches from E to random functions, which the adversary will not notice due to the assumed PRF security of E. Game G_2 switches to random core ciphertexts, which the adversary will not notice due to the assumed privacy of SE1. Since we can no longer assume that nonces are unique, however, the full ciphertexts may not be random. They will be, however, if the x values do not repeat, allowing us to switch to game G_3 with a loss that is the probability of such a repeat.

Proceeding to the details, assume as usual that A_2 does not make repeat or trivial queries.

Then we have

$$\begin{aligned}
&\mathbf{Adv}^{\mathrm{ae2}}_{\mathsf{SE}_{\mathrm{HN2}}}(A_2) = \Pr[\mathrm{G}_0(A_2)] - \Pr[\mathrm{G}_3(A_2)] \\
&= (\Pr[\mathrm{G}_0(A_2)] - \Pr[\mathrm{G}_1(A_2)]) \;+\; (\Pr[\mathrm{G}_1(A_2)] - \Pr[\mathrm{G}_2(A_2)]) \;+\; (\Pr[\mathrm{G}_2(A_2)] - \Pr[\mathrm{G}_3(A_2)]) \;.
\end{aligned}$$

To conclude the proof of Equation (1.9), we have

$$\begin{aligned}
\Pr[\mathrm{G}_0(A_2)] - \Pr[\mathrm{G}_1(A_2)] &= \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{E}}(B) \;, \\
\Pr[\mathrm{G}_1(A_2)] - \Pr[\mathrm{G}_2(A_2)] &= \mathbf{Adv}^{\mathrm{ae1}}_{\mathsf{SE1}}(A_1) \;, \\
\Pr[\mathrm{G}_2(A_2)] - \Pr[\mathrm{G}_3(A_2)] &\le \Pr[\mathrm{G}_3(A_2) \text{ sets } \mathsf{bad}] \\
&\le \frac{q_n q_e (q_e - 1)}{2^{\ell+1}} \;.
\end{aligned} \tag{1.10}$$

Adversaries B, A_1 for the first two equations above are those depicted in Fig. 1.9, and now $A_2 \in \mathcal{A}^{\mathrm{ae2}}_{\text{u-nmh}}$ because $A_1 \in \mathcal{A}^{\mathrm{ae1}}_{\text{u-nmh}}$. As before, we assume A_1 implements the f_i via lazy sampling. Games $\mathrm{G}_2, \mathrm{G}_3$ are identical-until-bad, so Equation (1.10) is by the Fundamental Lemma of Game Playing [30]. □

The above-sketched attack for the $\ell = 0$ case can be extended to an attack (adversary) that for arbitrary ℓ achieves an advantage of about $q_n q_e^2 \cdot 2^{-\ell}$, showing the bound of Theorem 8 is essentially tight. The idea is that the adversary can win when the ℓ stolen bits are the same across two ciphertexts encrypted to the same user. This extends an attack of [126] on Meyer-Matyas ciphertext stealing.

The result of Theorem 8, however, is not ideal, because security would need $\ell = 128$, which requires $\mathsf{SE1.mccl} \ge 128$ (not always true) and also, assuming 96-bit nonces, would require that the blockcipher E have block length 128+96=224, which precludes AES. We now give further transforms that do better.

1.7.3 The HN4 transform

The **HN3** transform clearly does *not* provide advanced-AE2-security because, if a nonce is repeated, the resulting ciphertexts have the same synthetic nonce, and hence the same first parts, which an adversary can notice. The starting idea for **HN4** is to obtain the synthetic nonce N_1 by applying the PRF F, not just to the actual nonce N as in **HN3**, but, as in SIV [125], to (N, M, H). If we now encrypt with N_1 under an NBE1 scheme $\mathsf{SE1}$, we can indeed show that AE2$[\mathcal{A}^{\text{ae2}}_{\text{u-nmh}}]$-security is achieved, assuming $\mathsf{SE1}$ is AE1$[\mathcal{A}^{\text{ae1}}_{\text{u-nmh}}]$-secure. The latter assumption, however, is not satisfactory here because AE1$[\mathcal{A}^{\text{ae1}}_{\text{u-nmh}}]$-security (typically achieved via SIV itself) already requires two passes through the entire input, so our computation of N_1 adds another entire pass, resulting in significant (non-constant) computational overhead. To avoid this we ask whether it would be enough for $\mathsf{SE1}$ to provide only privacy, meaning be AE1$[\mathcal{A}^{\text{ae1}}_{\text{r-n}} \cap \mathcal{A}^{\text{ae1}}_{\text{priv}}]$-secure, because this can be achieved in one pass. Indeed, this is what SIV assumes, but the difficulty is that SIV decryption makes crucial use of the original nonce N to provide authenticity, recomputing it and checking that it matches the one in the ciphertext. But to be nonce hiding, we cannot transmit N. We resolve this by including N as part of the message encrypted under $\mathsf{SE1}$.

Proceeding to the details, let $\mathsf{SE1}$ be an NBE1 scheme. Let F be a function family with $\mathsf{F.ol} = \mathsf{SE1.nl}$, meaning outputs of $\mathsf{F.Ev}$ can be used as nonces for $\mathsf{SE1}$, and also with $\mathsf{SE1.NS} \times \mathsf{SE1.MS} \times \mathsf{SE1.HS} \subseteq \mathsf{F.D}$, meaning triples (N, M, H) can be used as inputs to F. Let $\ell \geq 1$ be an integer prescribing the nonce length of the constructed scheme. Our **HN4** transform defines NBE2 scheme $\mathsf{SE}_{\text{HN4}} = \textbf{HN4}[\mathsf{SE1}, \ell, \mathsf{F}]$ whose encryption and decryption algorithms are shown in Figure 1.11. A key (K_{F}, K_1) for SE_{HN4} is a pair consisting of a key K_{F} for F and a key K_1 for $\mathsf{SE1}$, so that the key space is $\mathsf{SE}_{\text{HN4}}.\mathsf{KS} = \{0,1\}^{\mathsf{F.kl}} \times \mathsf{SE1.KS}$. The message and header spaces are unchanged, and the nonce space is $\mathsf{SE}_{\text{HN4}}.\mathsf{NS} = \{0,1\}^{\ell}$. The parsing in the second line of the decryption algorithm SE_{HN4} of Figure 1.5 is such that $|N_1| = \mathsf{SE1.nl}$. The ciphertext overhead is zero, and if $\mathsf{SE1}$ is a standard one-pass privacy only scheme like counter-mode, then the computational overhead is constant.

Security, as with SIV, requires that $\mathsf{SE1}$ satisfies tidiness [108]. Formally, for all K, N, C_1, H, if $\mathsf{SE1.Dec}(K,N,C_1,H) = M \neq \perp$ then $\mathsf{SE1.Enc}(K,N,M,H) = C_1$. Our assumption on $\mathsf{SE1}$ is AE1$[\mathcal{A}^{\text{ae1}}_{\text{r-n}} \cap \mathcal{A}^{\text{ae1}}_{\text{priv}}]$-security. (Privacy only, and again, for convenience, for random nonces.) By Theorem 3 this is implied by AE1$[\mathcal{A}^{\text{ae1}}_{\text{u-n}} \cap \mathcal{A}^{\text{ae1}}_{\text{priv}}]$-security. Assuming additionally that F is a PRF, the following says that $\mathbf{HN4}[\mathsf{SE1}, \ell, \mathsf{F}]$ is AE2$[\mathcal{A}^{\text{ae2}}_{\text{u-nmh}}]$-secure.

As we have often done before, we consider privacy and authenticity separately to show that the assumptions required, and bounds obtained, differ. Namely, assuming F is a PRF (1) privacy of $\mathsf{SE}_{\text{HN4}} = \mathbf{HN4}[\mathsf{SE1}, \ell, \mathsf{F}]$ is inherited from that of $\mathsf{SE1}$ with a tight reduction and (2) authenticity of SE_{HN4} assumes only the tidiness (not privacy) of $\mathsf{SE1}$.

Theorem 9 *Let* $\mathsf{SE}_{\text{HN4}} = \mathbf{HN4}[\mathsf{SE1}, \ell, \mathsf{F}]$ *be obtained as above, and assume* $\mathsf{SE1}$ *satisfies tidiness. Then, given adversary* $A_2 \in \mathcal{A}^{\text{ae2}}_{\text{u-nmh}} \cap \mathcal{A}^{\text{ae2}}_{\text{priv}}$ *making* q_n *queries to its* NEW *oracle and* q_e *queries per user to its* ENC *oracle, we construct adversaries* $A_1 \in \mathcal{A}^{\text{ae1}}_{\text{r-n}} \cap \mathcal{A}^{\text{ae1}}_{\text{priv}}$ *and* B_1 *such that*

$$\mathbf{Adv}^{\text{ae2}}_{\mathsf{SE2}}(A_2) \leq \mathbf{Adv}^{\text{prf}}_{\mathsf{F}}(B_1) + \mathbf{Adv}^{\text{ae1}}_{\mathsf{SE1}}(A_1)\ . \tag{1.11}$$

Adversary A_1 *preserves the resources of* A_2 *up to increasing the lengths of messages in* ENC *queries by* ℓ*. Adversary* B_1 *makes* q_n *queries to its* NEW *oracle, and* q_e *queries to its* FN *oracle per user, and its running time is about that of* A_2*. Also, given adversary* $A_2 \in \mathcal{A}^{\text{auth2}}_{\text{u-nmh}}$ *making* q_n *queries to its* NEW *oracle,* q_e *queries per user to its* ENC *oracle and* q_v *queries per user to its* VF *oracle, we construct adversary* B_2 *such that*

$$\mathbf{Adv}^{\text{auth2}}_{\mathsf{SE2}}(A_2) \leq \mathbf{Adv}^{\text{prf}}_{\mathsf{F}}(B_2) + \frac{q_n q_v}{2^{\mathsf{SE1.nl}}}\ . \tag{1.12}$$

Adversary B_2 *makes* q_n *queries to its* NEW *oracle, and* $q_e + q_v$ *queries per user to its* FN *oracle, and its running time is about that of* A_2*.*

Proof. For the proof of privacy, we will make use of the games $\mathrm{G}_0, \mathrm{G}_1, \mathrm{G}_2$ in Fig. 1.13. Game G_0 is the real game, game G_1 switches to using random functions, which the adversary

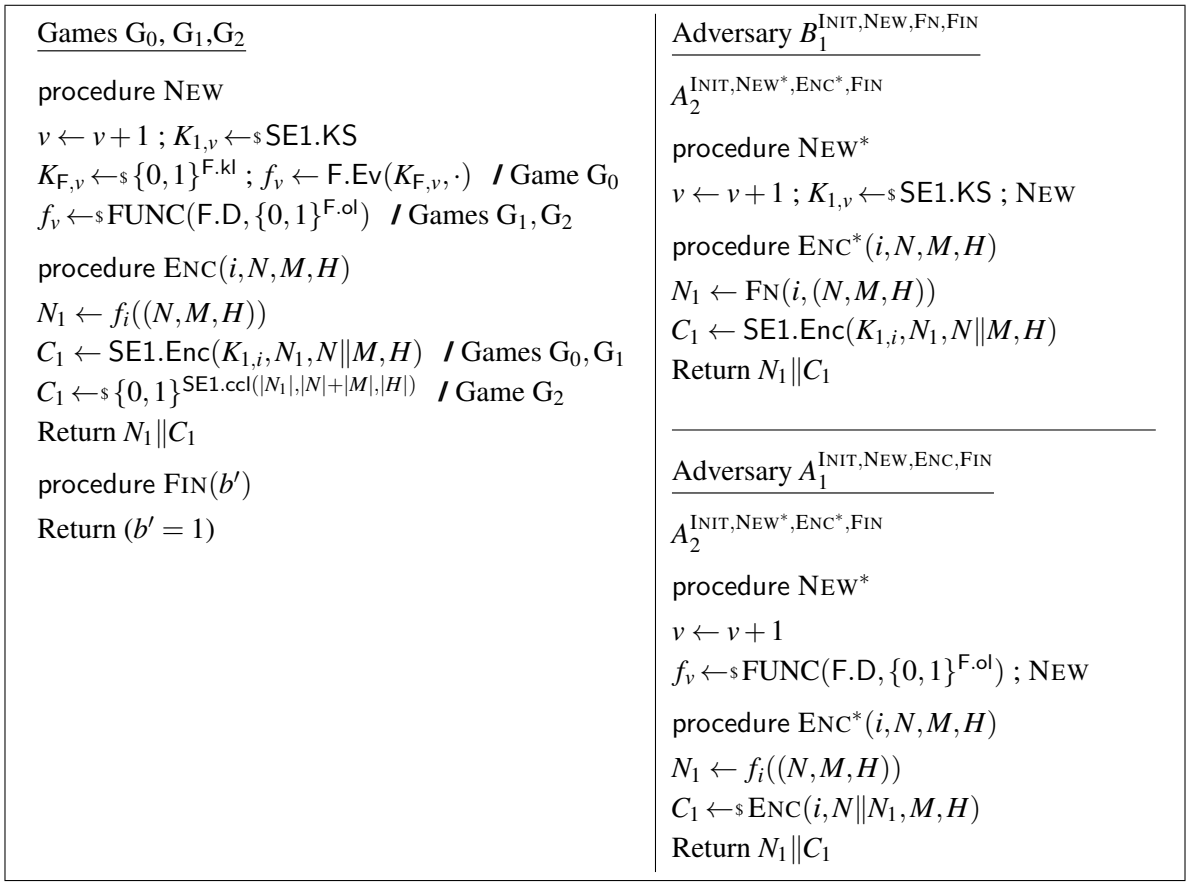

Figure 1.13. Games and adversaries used in proof of Equation (1.11). Note that $\mathsf{F.D} = \mathsf{SE1.NS} \times \mathsf{SE1.MS} \times \mathsf{SE1.HS}$, as required in th definition of **HN4** in Section 1.7.

Games $\boxed{\mathrm{G}_0}$, G_1

procedure NEW
$v \leftarrow v+1$; $K_{1,v} \leftarrow\$ \mathsf{SE1.KS}$
$K_{\mathsf{F},v} \leftarrow\$ \{0,1\}^{\mathsf{F.kl}}$; $f_v \leftarrow \mathsf{F.Ev}(K_{\mathsf{F},v},\cdot)$ // Game G_0
$f_v \leftarrow\$ \mathrm{FUNC}(\{0,1\}^{\mathsf{F.il}},\{0,1\}^{\mathsf{F.ol}})$ // Game G_1

procedure ENC(i,N,M,H)
$N_1 \leftarrow f_i((N,M,H))$
$C_1 \leftarrow \mathsf{SE1.Enc}(K_{1,i},N_1,N\|M,H)$
Return $N_1\|C_1$

procedure VF(i,C_2,H)
$N_1\|C_1 \leftarrow C_2$; $X \leftarrow \mathsf{SE1.Dec}(K_{1,i},N_1,C_1,H)$
If $(X=\perp)$ then return false
$N\|M \leftarrow X$; $T \leftarrow f_i((N,M,H))$
If $(T=N_1)$ then win $\leftarrow$ true
Return $(T=N_1)$

procedure FIN
Return win

Adversary $B_2^{\mathrm{INIT,NEW,FN,FIN}}$

$A_2^{\mathrm{INIT,NEW^*,ENC^*,VF^*,FIN^*}}$

procedure NEW*
$v \leftarrow v+1$; $K_{1,v} \leftarrow\$ \mathsf{SE1.KS}$; NEW

procedure ENC$^*(i,N,M,H)$
$N_1 \leftarrow \mathrm{FN}(i,(N,M,H))$
$C_1 \leftarrow \mathsf{SE1.Enc}(K_{1,i},N_1,N\|M,H)$
Return $N_1\|C_1$

procedure VF$^*(i,C_2,H)$
$N_1\|C_1 \leftarrow C_2$
$X \leftarrow \mathsf{SE1.Dec}(K_{1,i},N_1,C_1,H)$
If $(X=\perp)$ then return false
$N\|M \leftarrow X$; $T \leftarrow \mathrm{FN}(i,(N,M,H))$
If $(T=N_1)$ then win $\leftarrow$ true
Return $(T=N_1)$

procedure FIN*
If win = true then $b' \leftarrow 1$ else $b' \leftarrow 0$
FIN(b')

Figure 1.14. Games and adversaries used in proof of Equation (1.12).

will not notice due to the assumed PRF security of F, and game G_2 switches to random core ciphertexts. Adversaries B_2, A_1 are also depicted in Fig. 1.13. Adversary A_2, being a privacy adversary, makes no DEC queries, so we omit giving oracle DEC in the games as well as when it is run by other adversaries. As usual, we assume that A_1 implements f_i using lazy sampling for efficiency. Because we assumed the nonce-message-header triples provided to f_i by A_2 do not repeat, G_2 has random full ciphertexts and $A_1 \in \mathcal{A}^{\mathrm{ae1}}_{\mathrm{r\text{-}n}}$. From here, we can derive Equation 1.11:

$$\begin{aligned}
&\mathbf{Adv}^{\mathrm{ae2}}_{\mathsf{SE}_{\mathrm{HN4}}}(A_2) = \Pr[\mathrm{G}_0(A_2)] - \Pr[\mathrm{G}_2(A_2)] \\
&= (\Pr[\mathrm{G}_0(A_2)] - \Pr[\mathrm{G}_1(A_2)]) \;+\; (\Pr[\mathrm{G}_1(A_2)] - \Pr[\mathrm{G}_2(A_2)]) \\
&= \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(B) \;+\; \mathbf{Adv}^{\mathrm{ae1}}_{\mathsf{SE}_{\mathsf{SE1}}}(A_1)\;.
\end{aligned}$$

Now we proceed to the authenticity proof. As before, we assume that A_2 does not make repeat or trivial queries. Games $\mathrm{G}_0, \mathrm{G}_1$ and adversary B_2 are depicted in Fig. 1.14. As before, the difference is that G_1 switches the f_v functions to random. We have

$$\begin{aligned}\mathbf{Adv}^{\mathrm{auth2}}_{\mathsf{SE}_{\mathrm{HN4}}}(A_2) &= \Pr[\mathrm{G}_0(A_2)] \\ &= \Pr[\mathrm{G}_1(A_2)] + (\Pr[\mathrm{G}_0(A_2)] - \Pr[\mathrm{G}_1(A_2)]) \ .\end{aligned}$$

To complete the proof, we claim that

$$\begin{aligned}\Pr[\mathrm{G}_0(A_2)] - \Pr[\mathrm{G}_1(A_2)] &\le \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(B_2) \\ \Pr[\mathrm{G}_1(A_2)] &\le \frac{q_n q_v}{2^{\mathsf{SE1.nl}}} \ . \end{aligned} \tag{1.13}$$

Equation (1.13) is due to the assumed tidiness of $\mathsf{SE1}$, as follows. Suppose $\perp \neq X = N \| M$. Tidiness plus the assumption that A_2 makes no trivial queries say that (i, N, M, H) was not a prior query to ENC, which means that $T = N_1$ with probability at most $2^{-\mathsf{SE1.nl}}$. $\square$

1.7.4 The HN5 transform

Our final transform **HN5** is different. It does not start from an NBE1 scheme but rather from a (arbitrary-input-length) tweakable cipher, extending the encode-then-encipher paradigm [29] to provide advanced-AE2-security. Instantiation via a fast tweakable cipher like AEZ [80] results in correspondingly fast advanced-AE2-secure NBE2.

We encipher the nonce, message and some redundancy, using the header as the tweak. The change from [80] is to move the nonce from tweak to an input so as to hide it, which we will show is enough to confer AE2-security.

Tweakable ciphers.

These are the basic tool for this transform, so we recall definitions. A tweakable cipher TE [102, 80] specifies a deterministic evaluation algorithm $\mathsf{TE.Ev} : \{0,1\}^{\mathsf{TE.kl}} \times \mathsf{TE.TS} \times$

Game $\mathbf{G}^{\text{prf}}_{\mathsf{TE}}$	Game $\mathbf{G}^{\text{prp-cca}}_{\mathsf{TE}}$
procedure INIT	procedure INIT
$b \leftarrow\!\!\$\ \{0,1\}$	$b \leftarrow\!\!\$\ \{0,1\}$
procedure NEW	procedure NEW
$v \leftarrow v+1$	$v \leftarrow v+1$
If $b=1$ then	If $b=1$ then
$\quad K_v \leftarrow\!\!\$\ \{0,1\}^{\mathsf{TE.kl}}$	$\quad K_v \leftarrow\!\!\$\ \{0,1\}^{\mathsf{TE.kl}}$
$\quad$ For all $T \in \mathsf{TE.TS}$ do	$\quad$ For all $T \in \mathsf{TE.TS}$ do $\pi_{v,T} \leftarrow \mathsf{TE.Ev}(K_v, T, \cdot)$
$\quad\quad f_{v,T} \leftarrow \mathsf{TE.Ev}(K_v, T, \cdot)$	Else For all $T \in \mathsf{TE.TS}$ do $\pi_{v,T} \leftarrow\!\!\$\ \text{LPERM}$
Else For all $T \in \mathsf{TE.TS}$ do	procedure $\text{FN}(i,T,X)$
$\quad f_{v,T} \leftarrow\!\!\$\ \text{LFUNC}$	Return $\pi_{v,T}(X)$
procedure $\text{FN}(i,T,X)$	procedure $\text{FN}^{-1}(i,T,Y)$
Return $f_{i,T}(X)$	Return $\pi^{-1}_{i,T}(Y)$
procedure $\text{FIN}(b')$	procedure $\text{FIN}(b')$
Return $(b=b')$	Return $(b=b')$

Figure 1.15. Game defining (multi-user) PRF security for tweakable cipher TE (left) and game defining (multi-user) PRP-CCA security for TE (right).

$\{0,1\}^* \to \{0,1\}^*$ and a deterministic inversion algorithm $\mathsf{TE.In} : \{0,1\}^{\mathsf{TE.kl}} \times \mathsf{TE.TS} \times \{0,1\}^* \to \{0,1\}^*$. Here, $\mathsf{TE.kl}$ is the key length and $\mathsf{TE.TS}$ is the tweak space. We require that for all $K \in \{0,1\}^{\mathsf{TE.kl}}$, $T \in \mathsf{TE.TS}$ and $X \in \{0,1\}^*$ we have $|\mathsf{TE.Ev}(K,T,X)| = |X|$ and $\mathsf{TE.In}(K,T, \mathsf{TE.Ev}(K,T,X)) = X$.

We define (multi-user) PRF security for tweakable cipher TE via the game $\mathbf{G}^{\text{prf}}_{\mathsf{TE}}(A)$ in Fig. 1.15. Here LFUNC is the set of all length-preserving functions $f : \{0,1\}^* \to \{0,1\}^*$. It is required that any $\text{FN}(i,T,X)$ query of the adversary A satisfies $i \le v$, $T \in \mathsf{TE.TS}$ and $X \in \{0,1\}^*$. The (multi-user) PRF advantage of A is $\mathbf{Adv}^{\text{prf}}_{\mathsf{TE}}(A) = 2\Pr[\mathbf{G}^{\text{prf}}_{\mathsf{TE}}(A)] - 1$

We define (multi-user) PRP-CCA security [102] for tweakable cipher TE via the game $\mathbf{G}^{\text{prp-cca}}_{\mathsf{TE}}(A)$ in Fig. 1.15. Here LPERM is the set of all length-preserving bijections $\pi : \{0,1\}^* \to \{0,1\}^*$. (Note that for any such π and any n, restricting π to $\{0,1\}^n$ yields a permutation on $\{0,1\}^n$.) It is required that any $\text{FN}(i,T,X)$ or $\text{FN}^{-1}(i,T,Y)$ query of adversary A satisfies $i \le v$, $T \in \mathsf{TE.TS}$ and $X, Y \in \{0,1\}^*$. The (multi-user) PRP-CCA advantage of A is $\mathbf{Adv}^{\text{prp-cca}}_{\mathsf{TE}}(A) =$

$2\Pr[\mathbf{G}^{\text{prp-cca}}_{\mathsf{TE}}(A)] - 1$.

The HN5 transform.

Proceeding to the details, let TE be a tweakable cipher as defined in Section 1.2. Let $\ell \geq 1$ be an integer prescribing the nonce length of the constructed scheme. Let $\ell_z \geq 0$ be the number of bits of redundancy we introduce to provide authenticity [29]. Our transform defines NBE2 scheme $\mathsf{SE}_{\text{HN5}} = \mathbf{HN5}[\mathsf{TE}, \ell, \ell_z]$ whose encryption and decryption algorithms are shown in Figure 1.11. The key space of SE_{HN5} is the key space of TE. The message space is $\{0,1\}^*$. The header space $\mathsf{SE}_{\text{HN5}}.\mathsf{HS}$ is set to the tweak space $\mathsf{TE}.\mathsf{TS}$ of TE. The nonce space is $\mathsf{SE}_{\text{HN5}}.\mathsf{NS} = \{0,1\}^{\ell}$. The length of ciphertext $\mathsf{SE}_{\text{HN5}}.\mathsf{Enc}(K,N,M,H)$ is $\ell_z + |N| + |M|$, so $\mathsf{SE}_{\text{HN5}}.\mathsf{CS}(\ell_n, \ell_m, \ell_h) = \{0,1\}^{\ell_z+\ell+\ell_m}$. Ciphertext overhead, in this case, is not relative to an underlying NBE1 scheme, since there isn't any, but we see that ciphertexts are longer than message plus nonce by just ℓ_z bits, which is effectively optimal [80].

With this transform, it is helpful to establish privacy and authenticity separately because the security notions required to tightly bound them differ. The privacy of SE_{HN5} reduces to the PRF security of TE while its authenticity depends on TE being an PRP-CCA secure tweakable cipher and ℓ_z being sufficiently large. The following theorem captures this formally.

Theorem 10 *Let $\mathsf{SE}_{\text{HN5}} = \mathbf{HN5}[\mathsf{TE}, \ell, \ell_z]$ be obtained as above. Then, given adversary $A_2 \in \mathcal{A}^{\text{ae2}}_{\text{u-nmh}} \cap \mathcal{A}^{\text{ae2}}_{\text{priv}}$, making q_n queries to its* NEW *oracle and q_e queries per user to its* ENC *oracle, we construct adversary B_1 such that*

$$\mathbf{Adv}^{\text{ae2}}_{\mathsf{SE}_{\text{HN5}}}(A) \;\leq\; \mathbf{Adv}^{\text{prf}}_{\mathsf{TE}}(B_1)\,. \tag{1.14}$$

Adversary B makes q_n queries to its NEW *oracle and q_e queries per user to its* FN *oracle, and its running time is about that of A. Also, given adversary $A_2 \in \mathcal{A}^{\text{auth2}}_{\text{u-nmh}}$ making q_n queries to its* NEW *oracle and q_e, q_v queries per user to its* ENC, VF *oracles respectively, with $q_e + q_v \leq 2^{\ell+\ell_z-1}$, we*

Adversary $B_1^{\text{INIT,NEW,FN,FIN}}$	Adversary $B_2^{\text{INIT,NEW,FN,FN}^{-1}\text{,FIN}}$
$A^{\text{INIT,NEW,ENC}^*\text{,FIN}}$	INIT $A_2^{\text{NEW,ENC}^*\text{,VF}^*\text{,FIN}^*}$
procedure $\text{ENC}^*(i,N,M,H)$	procedure $\text{ENC}^*(i,N,M,H)$
$C_2 \leftarrow \text{FN}(i,H,0^{\ell_z}\|N\|M)$	$C_2 \leftarrow \text{FN}(i,H,0^{\ell_z}\|N\|M)$; Return C_2
Return C_2	procedure $\text{VF}^*(i,C_2,H)$
	$X \leftarrow \text{FN}^{-1}(i,H,C_2)$
	If $(X[1..\ell_z] \neq 0^{\ell_z})$ then return false
	Else win $\leftarrow$ true ; Return true
	procedure FIN^*
	If $(\text{win} = \text{true})$ then $b' \leftarrow 1$ else $b' \leftarrow 0$
	$\text{FIN}(b')$

Figure 1.16. Adversaries used in the proof of Theorem 10.

construct adversary B_2 such that

$$\mathbf{Adv}^{\text{auth2}}_{\mathsf{SE}_{\text{HN5}}}(A_2) \;\leq\; \mathbf{Adv}^{\text{prp-cca}}_{\mathsf{TE}}(B_2) + \frac{2q_nq_d}{2^{\ell_z}} \;. \tag{1.15}$$

Adversary B_2 makes q_n queries to its NEW *oracle and q_e, q_v queries per user to its* $\text{FN}, \text{FN}^{-1}$ *oracles respectively, and its running time is about that of A_2.*

Proof. Adversary B_1 referred to in Equation (1.14) is in Fig. 1.16. INIT, FIN and NEW are all unchanged, and ENC is simulated as shown. Since A is a privacy adversary, we do not need to simulate a decryption oracle.

Adversary B_2 referred to in Equation (1.15) is also presented in Fig. 1.16. As before, we assume A_2 neither makes repeat encryption or verification queries, nor makes trivial verification queries, meaning it does not make query $\text{VF}(i,C_2,H)$ if it has previously received C_2 in response to an $\text{ENC}(i,\cdot,\cdot,H)$ query and also $|C_2| \geq \ell + \ell_z$ in any $\text{VF}(i,C_2,H)$ query.

Let b be the challenge bit of game $\mathbf{G}^{\text{prp-cca}}_{\mathsf{TE}}$ and let b' be the bit that B_2 queries to

$\mathbf{G}^{\text{prp-cca}}_{\mathsf{TE}}$.FIN. Then,

$$\mathbf{Adv}^{\text{prp-cca}}_{\mathsf{TE}}(B_2) = \Pr[b' = 1 \,|\, b = 1] - \Pr[b' = 1 \,|\, b = 0] \,.$$

To complete the proof, we claim that

$$\Pr[b' = 1 \,|\, b = 1] \geq \mathbf{Adv}^{\text{auth2}}_{\mathsf{SE}_{\text{HN5}}}(A_2) \tag{1.16}$$

$$\Pr[b' = 1 \,|\, b = 0] \leq \frac{2q_n q_v}{2^{\ell_z}} \,. \tag{1.17}$$

Note that $b' = 1$ if and only if some query of A_2 to VF* returns true. If $b = 1$ then this happens if A_2 wins $\mathbf{G}^{\text{auth2}}_{\mathsf{SE}_{\text{HN5}}}$, justifying Equation (1.16). Now suppose $b = 0$. Consider a particular user i and the j-th VF query to that user. Let C_2 be the ciphertext in that query and assume s queries to ENC have been made to user i prior to this VF query. Then the probability that this VF query sets win to true is at most

$$\begin{aligned}\frac{2^{|C_2|-\ell_z} - s}{2^{|C_2|} - (s+j-1)} &\leq \frac{2^{|C_2|-\ell_z}}{2^{|C_2|} - (q_e+q_v-1)} \\ &= \frac{1}{2^{\ell_z}} \cdot \frac{1}{1-(q_e+q_v-1)\cdot 2^{-|C_2|}} \,.\end{aligned}$$

But $|C_2| \geq \ell + \ell_z$ for any ciphertext C_2 in a VF query, and we assumed $q_e + q_v \leq 2^{\ell+\ell_z-1}$, so, across all queries, the probability that win is set to true is at most

$$\begin{aligned}\frac{q_n q_v}{2^{\ell_z}} \cdot \frac{1}{1-(q_e+q_v-1)\cdot 2^{-(\ell+\ell_z)}} &\leq \frac{q_n q_v}{2^{\ell_z}} \cdot \frac{1}{1-2^{(\ell+\ell_z-1)}\cdot 2^{-(\ell+\ell_z)}} \\ &= \frac{q_n q_v}{2^{\ell_z}} \cdot \frac{1}{1-2^{-1}} \,,\end{aligned}$$

□

1.8 Dedicated transform for GCM

We have shown that our generic transforms allow us to immunize NBE1 schemes with low overhead. We now present a transform specific to the GCM NBE1 scheme which is used in TLS. Our transform takes advantage of the underlying structure of GCM to further minimize overhead. We also minimize changes to the scheme so that existing hardware and software can easily adapt.

Padding function.

Let $\pi : \{0,1\}^n \to \{0,1\}^n$ be a function. (In the scheme it will be $\mathsf{E.Ev}(K,\cdot)$ for a blockcipher E.) We want to run it in counter mode, defining a function $\mathrm{Pad}^{\pi}_{s,t}$ that takes a nonce $N \in \{0,1\}^*$ of length at most n to return a string (the pad) of length t, with t not necessarily a multiple of n. Integer $s \geq 0$ is the starting point. Recall that if i is an integer then as per Section 1.2, $\langle i \rangle_m$ is the m-bit representation of i mod 2^m. Now we can define:

$\underline{\mathrm{Pad}^{\pi}_{s,t}(N)}$

$L \leftarrow \lfloor t/n \rfloor$; $e \leftarrow t - nL$; $X \leftarrow \varepsilon$

For $i = 0, \ldots, L-1$ do $X \leftarrow X \,\|\, \pi(N \| \langle s+i \rangle_{n-|N|})$

$X \leftarrow X \,\|\, \pi(N \| \langle s+L \rangle_{n-|N|})[1..e]$

Return X

The CAU1 transform.

Following [31], we generalize GCM via a transform **CAU1**. (We add the "1" to indicate that it is an NBE1 scheme.) Let E be a blockcipher. Let H be a function family with $\mathsf{H.D} = \{0,1\}^* \times \{0,1\}^*$ and $\mathsf{H.ol} = \mathsf{H.kl} = \mathsf{E.bl}$. Let $1 \leq \ell < \mathsf{E.bl}$ be an integer indicating the nonce-length. We associate to these the NBE1 scheme $\mathsf{SE1} = \mathbf{CAU1}[\mathsf{E}, \mathsf{H}, \ell]$ whose encryption and decryption algorithms are shown at the top of Fig. 1.17. The key K is a key for E, meaning $\mathsf{SE1.KS} = \{0,1\}^{\mathsf{E.kl}}$. The header space is $\mathsf{SE1.HS} = \{0,1\}^*$. The message space $\mathsf{SE1.MS}$ is the

$\underline{\mathsf{SE1.Enc}(K,N,M,H)}$
$P \leftarrow \mathrm{Pad}^{\mathsf{E.Ev}(K,\cdot)}_{2,|M|}(N)$
$C_1^* \leftarrow M \oplus P$
$K_{\mathsf{H}} \leftarrow \mathsf{E.Ev}(K, 0^{\mathsf{E.bl}})$
$h \leftarrow \mathsf{H.Ev}(K_{\mathsf{H}}, (C_1^*, H))$
$\tau \leftarrow h \oplus \mathsf{E.Ev}(K, N \| \langle 1 \rangle_{\mathsf{E.bl}-\ell})$
$C_1 \leftarrow \tau \| C_1^*$; Return C_1

$\underline{\mathsf{SE1.Dec}(K,N,C_1,H)}$
$\tau \| C_1^* \leftarrow C_1$; $P \leftarrow \mathrm{Pad}^{\mathsf{E.Ev}(K,\cdot)}_{2,|C_1^*|}(N)$
$M \leftarrow C_1^* \oplus P$
$K_{\mathsf{H}} \leftarrow \mathsf{E.Ev}(K, 0^{\mathsf{E.bl}})$
$h \leftarrow \mathsf{H.Ev}(K_{\mathsf{H}}, (C_1^*, H))$
$\tau' \leftarrow h \oplus \mathsf{E.Ev}(K, N \| \langle 1 \rangle_{\mathsf{E.bl}-\ell})$
If $(\tau = \tau')$ then return M else return $\perp$

$\underline{\mathsf{SE2.Enc}(K,N,M,H)}$
$C_2 \leftarrow \mathsf{SE1.Enc}(K, N, 0^{\ell} \| M, H)$
Return C_2

$\underline{\mathsf{SE2.Dec}(K,C_2,H)}$
$\tau \| C_1^* \leftarrow C_2$; $K_{\mathsf{H}} \leftarrow \mathsf{E.Ev}(K, 0^{\mathsf{E.bl}})$; $h \leftarrow \mathsf{H.Ev}(K_{\mathsf{H}}, (C_1^*, H))$
$y \leftarrow \mathsf{E.In}(K, \tau \oplus h)$; $N \| w \leftarrow y$
$P \leftarrow \mathrm{Pad}^{\mathsf{E.Ev}(K,\cdot)}_{2,|C_1^*|}(N)$; $M^* \leftarrow C_1^* \oplus P$; $x \| M \leftarrow M^*$
If $((x = 0^{\ell})$ and $(w = \langle 1 \rangle_{\mathsf{E.bl}-\ell}))$ then return M else return $\perp$

Figure 1.17. Encryption and decryption algorithms of NBE1 scheme $\mathsf{SE1} = \mathbf{CAU1}[\mathsf{E}, \mathsf{H}, \ell]$ and NBE2 scheme $\mathsf{SE2} = \mathbf{CAU2}[\mathsf{E}, \mathsf{H}, \ell]$. SE2's encryption algorithm uses that of SE1 as a subroutine.

set of strings of length at most $\mathsf{E.bl} \cdot (2^{\mathsf{E.bl}-\ell} - 2)$. The nonce space is $\mathsf{SE1.NS} = \{0,1\}^{\ell}$. In the pseudocode of Fig. 1.17, the parsing $\tau \| C_1^* \leftarrow C_1$ is such that $|\tau| = \mathsf{E.bl}$, and if parsing fails it is understood that the algorithm returns $\perp$.

AES-GCM, as proposed by McGrew and Viega [105] and standardized by NIST [61], is obtained by setting $\mathsf{E} = \mathsf{AES}$ (so $\mathsf{E.bl} = 128$), $\mathsf{H} = \mathsf{GHASH}$ and $\ell = 96$. It is widely used in practice and proven to provide basic AE1-security (i.e. $\mathrm{AE1}[\mathcal{A}^{\mathrm{ae2}}_{\mathrm{u\text{-}n}}]$-security). SE1 has a fixed-length nonce, reflecting the standardized version of GCM, but a variant with variable-length nonces can be obtained by pre-processing the nonce, as discussed in [105, 84].

Our CAU2 transform.

To provide nonce hiding security, we exploit a feature of NBE1 scheme $\mathsf{SE1} = \mathbf{CAU1}[\mathsf{E}, \mathsf{H}, \ell]$, namely that the nonce can be obtained from the authentication tag τ. In particular, if $\tau \| C_1^* \leftarrow \mathsf{SE1.Enc}(K,N,M,H)$ and $K_{\mathsf{H}} = \mathsf{E.Ev}(K, 0^{\mathsf{E.bl}})$ then the nonce N can be recovered as

the first ℓ bits of

$$y = \mathsf{E.In}(K, \tau \oplus \mathsf{H.Ev}(K_{\mathsf{H}}, (C_1^*, H))) \ .$$

Therefore, in our NBE2 variant $\mathsf{SE2} = \mathbf{CAU2}[\mathsf{E}, \mathsf{H}, \ell]$, we don't explicitly communicate the nonce but rather have the receiver use the tag to compute y as above, rejecting if the last $\mathsf{E.bl} - \ell$ bits of y are not $\langle 1 \rangle_{\mathsf{E.bl}-\ell}$ and otherwise setting N to the first ℓ bits of y. This can be seen as exploiting the "parsimoniousness" of $\mathbf{TN}[\mathsf{SE1}]$ [28]. Unfortunately, merely doing this results in a loss of authenticity because the decryption procedure will succeed for any given ciphertext with probability $2^{-\mathsf{E.bl}+\ell}$, since this is the probability that *some* nonce with suffix $\langle 1 \rangle_{\mathsf{E.bl}-\ell}$ is recovered. This would be unacceptable in GCM since an adversary would be able to forge valid ciphertexts with probability 2^{-32}. So in order to retain security, we add redundancy to the message before encrypting, specifically prepending it with 0^ℓ. Decryption will check that the message returned by $\mathsf{SE1.Dec}$ indeed starts with such a string of 0s. We expect that decryption with a "wrong" nonce leads to a ciphertext that lacks the redundancy. A similar technique is used by ADL [13] in their scheme, GCM-RUP, but for a slightly different variant of GCM.

More formally, let $\mathsf{E}, \mathsf{H}, \ell$ be as for **CAU1** above. Our transform **CAU2** defines an NBE2 scheme $\mathsf{SE2} = \mathbf{CAU2}[\mathsf{E}, \mathsf{H}, \ell]$ whose encryption and decryption algorithms are shown at the bottom of Fig. 1.17. The key, header and nonce spaces are the same as for $\mathsf{SE1} = \mathbf{CAU1}[\mathsf{E}, \mathsf{H}, \ell]$. To allow room for the redundancy, the maximum message length is reduced by ℓ bits, so the message space is the set of all strings of length at most $\mathsf{E.bl} \cdot (2^{\mathsf{E.bl}-\ell} - 2) - \ell$. In the pseudocode of Fig. 1.17, the parsing $N \| w \leftarrow y$ is such that $|N| = \ell$ and $|w| = \mathsf{E.bl} - \ell$. The parsing $x \| M \leftarrow M^*$ is such that $|x| = \ell$, and if parsing fails it is understood that the algorithm returns $\perp$.

Of course an AE2-secure $\mathbf{CAU2}[\mathsf{E}, \mathsf{H}, \ell]$ scheme could be obtained from $\mathbf{CAU1}[\mathsf{E}, \mathsf{H}, \ell]$ via our basic transforms of Section 1.6, but $\mathbf{CAU2}[\mathsf{E}, \mathsf{H}, \ell]$ has the following advantages over these schemes. It does not change the key, adding no new key materiel. For encryption the code of $\mathbf{CAU1}[\mathsf{E}, \mathsf{H}, \ell]$ can be invoked in a blackbox way, so existing (often extensively optimized) implementations may be reused and existing hardware and software can more easily adapt.

Decryption, however, requires more extensive implementation changes.

In the following, we establish basic AE2 security of $\mathbf{CAU2}[\mathsf{E},\mathsf{H},\ell]$ assuming PRF-security of E and AXU-security of H. This result improves on the one claimed in the preliminary version of our paper [27], which had needed the stronger assumption that E is a strong PRP. (Meaning, a PRP when the adversary can query both the function and its inverse.) Theorem 1 allows us to consider privacy and authenticity separately. As Theorem 11 below indicates, privacy is trivially inherited from $\mathbf{CAU1}[\mathsf{E},\mathsf{H},\ell]$. The proof for authenticity, namely that of Theorem 12, is more invasive and non-trivial.

Privacy of $\mathbf{CAU2}[\mathsf{E},\mathsf{H},\ell]$.

For privacy of a scheme, only the encryption algorithm is relevant; how decryption is performed makes no difference. Now, as Figure 1.17 indicates, the encryption algorithm of $\mathsf{SE2} = \mathbf{CAU2}[\mathsf{E},\mathsf{H},\ell]$ simply runs that of $\mathsf{SE1} = \mathbf{CAU1}[\mathsf{E},\mathsf{H},\ell]$ with 0^ℓ prepended to the message. As a result, privacy of $\mathsf{SE2}$ follows directly from that of $\mathsf{SE1}$:

Theorem 11 *Let* $\mathsf{SE1} = \mathbf{CAU1}[\mathsf{E},\mathsf{H},\ell]$ *and* $\mathsf{SE2} = \mathbf{CAU2}[\mathsf{E},\mathsf{H},\ell]$ *be obtained as above. Then, given adversary* $A_2 \in \mathcal{A}^{\text{ae2}}_{\text{priv}} \cap \mathcal{A}^{\text{ae2}}_{\text{u-n}}$ *we construct* $A_1 \in \mathcal{A}^{\text{ae1}}_{\text{priv}} \cap \mathcal{A}^{\text{ae1}}_{\text{u-n}}$ *such that*

$$\mathbf{Adv}^{\text{ae2}}_{\mathsf{SE2}}(A_2) \;\le\; \mathbf{Adv}^{\text{ae1}}_{\mathsf{SE1}}(A_1)\,. \tag{1.18}$$

Adversary A_1 *preserves the resources of* A_2 *up to an increase of* ℓ *in the lengths of any messages queried to* ENC.

Proof. When A_2 makes a query (i,N,M,H) to its encryption oracle, A_1 queries $(i,N,0^\ell\|M,H)$ to its encryption oracle and returns the result to A_2. Since these are privacy adversaries, there are no decryption queries to consider. When A_2 makes its query to its FIN oracle, adversary A_1 makes the same query to its own FIN oracle. □

This allows us to conclude privacy of $\mathsf{SE2} = \mathbf{CAU2}[\mathsf{E},\mathsf{H},\ell]$ based on known proofs and bounds for $\mathsf{SE1} = \mathbf{CAU1}[\mathsf{E},\mathsf{H},\ell]$ from prior work [105, 84, 31, 103, 82]. In particular this allows $\mathsf{SE2}$

Game $\mathbf{G}^{\mathrm{axu}}_{\mathsf{H}}$	procedure $\textsc{Fin}((x_1,y_1),(x_2,y_2),z)$
procedure $\textsc{Init}$	$h_1 \leftarrow \mathsf{H.Ev}(L,(x_1,y_1))$; $h_2 \leftarrow \mathsf{H.Ev}(L,(x_2,y_2))$
$L \xleftarrow{\$} \{0,1\}^{\mathsf{H.kl}}$	Return $((h_1 \oplus h_2 = z)$ and $((x_1,y_1) \neq (x_2,y_2)))$

Figure 1.18. Game defining AXU security for function family H.

to inherit the high-quality bounds shown for $\mathsf{SE1}$ shown by Hoang, Tessaro and Thiruvengadam [82].

AXU security.

The authenticity of $\mathsf{SE2} = \mathbf{CAU2}[\mathsf{E},\mathsf{H},\ell]$ assumes axu security of H. We will define a weaker, computational version of the usually information-theoretic definition of [100, 98, 5, 31], and show that this suffices, which makes our results stronger.

Let H be a function family with $\mathsf{H.D} = \{0,1\}^* \times \{0,1\}^*$. Consider game $\mathbf{G}^{\mathrm{axu}}_{\mathsf{H}}$ of Figure 1.18, and let C be an adversary, that we call an axu-adversary, playing this game. Note that the key L chosen in $\textsc{Init}$ is not returned to the adversary. The adversary has no oracles. To win, it must find, and submit to $\textsc{Fin}$, a pair $(x_1,y_1),(x_2,y_2)$ of distinct messages, together with the value z of the xor of $\mathsf{H.Ev}(L,\cdot)$ on these messages. We let $\mathbf{Adv}^{\mathrm{axu}}_{\mathsf{H}}(C) = \Pr[\mathbf{G}^{\mathrm{axu}}_{\mathsf{H}}(C)]$ be the probability that the adversary wins.

The advantage of C will depend on the lengths of the inputs in its $\textsc{Fin}$ query. These are accordingly quantified in Theorem 12. The computational element of this AXU treatment is that Theorem 12 constructs an adversary C with bounded (and specified) resources.

The AXU family GHASH underlying GCM fits in our framework, so our results apply to it. But, unlike prior results, ours apply to other families as well. For example, we could set H to be a PRF or a collision-resistant hash function like SHA256, choices whose security is only computational.

Authenticity of $\mathbf{CAU2}[\mathsf{E},\mathsf{H},\ell]$.

We exploit our general results to reduce to as simple a case as possible. (Better bounds

may be possible by direct approaches.) First, Theorem 2 allows us to restrict attention to a single user. Now, still with a single user, Theorem 1 allows us to bound the auth2 advantage for adversaries that are orderly. Finally, a trivial hybrid argument says that, for orderly adversaries, we can assume just one VF query. Thus, below, the given adversary A_2 against $\mathsf{SE2} = \mathbf{CAU2}[\mathsf{E},\mathsf{H},\ell]$ is assumed to be orderly, to make one NEW query (single user) and to make one VF query.

The proof of this result is non-trivial because the natural approach to this proof is to begin by switching $\mathsf{E.Ev}(K,\cdot)$ to a random permutation. This would need us to assume prp-cca (strong prp) security because the inverse function is computed in VF. Instead our proof delays the switch, staying with $\mathsf{E.Ev}(K,\cdot)$ and exploiting its being a permutation to move to a game in which VF does not need to compute the inverse $\mathsf{E.In}(K,\cdot)$. Once this is done, we can switch $\mathsf{E.Ev}(K,\cdot)$ to a random *function* and rely only on the PRF assumption. Then, another game sequences is used to reduce to the assumed axu-security of H.

Theorem 12 *Let* $\mathsf{SE2} = \mathbf{CAU2}[\mathsf{E},\mathsf{H},\ell]$ *be obtained as above. Then, given adversary* $A_2 \in \mathcal{A}^{\text{auth2}}_{\text{u-n}} \cap \mathcal{A}^{\text{auth2}}_{\text{ord}}$ *making one query to its* NEW *oracle,* q_e *queries to its* ENC *oracle and one query to its* VF *oracle, we construct adversaries* B, C *such that*

$$\mathbf{Adv}^{\text{auth2}}_{\mathsf{SE2}}(A_2) \le 2 \cdot \mathbf{Adv}^{\text{prf}}_{\mathsf{E}}(B) + q_e \cdot \mathbf{Adv}^{\text{axu}}_{\mathsf{H}}(C) + \frac{1}{2^\ell}.$$

Let σ *be the total number of blocks across the messages queried by* A_2 *to* ENC. *Let* m *be the maximum, over all these queries, of the length of the message plus the length of the header in the query. Let* m' *be the length of the ciphertext plus the length of the header in the* VF *query. Then adversary* B *makes* $\sigma + q_e$ *queries to its* FN *oracle and its running time is about that of* A_2. *The messages submitted by* C *to* FIN *have lengths at most* $\max(m + \mathsf{E.bl}, m')$ *and the running time of* C *is about that of* A_2.

Games G_0, $\boxed{\mathrm{G}_1}$, G_2, G_3

procedure NEW
$K \leftarrow\!\!\$\ \{0,1\}^{\mathsf{E.bl}}$; $\pi \leftarrow \mathsf{E.Ev}(K,\cdot)$ // Games $\mathrm{G}_0, \mathrm{G}_1, \mathrm{G}_2$
$\pi \leftarrow\!\!\$\ \mathrm{FUNC}(\{0,1\}^{\mathsf{E.bl}})$ // Game G_3
$L \leftarrow \pi(0^{\mathsf{E.bl}})$

procedure ENC$(1, N, M, H)$
$i \leftarrow i+1$; $N_i \leftarrow N$; $M_i \leftarrow M$; $H_i \leftarrow H$; $P_i \leftarrow \mathrm{Pad}^{\pi}_{2,\ell+|M|}(N)$; $M^* \leftarrow 0^{\ell} \| M$; $C^*_{1,i} \leftarrow M \oplus P_i$
$h_i \leftarrow \mathsf{H.Ev}(L, (C^*_{1,i}, H_i))$; $p_i \leftarrow \pi(N_i \| \langle 1 \rangle_{\mathsf{E.bl}-\ell})$; $\mathcal{P} \leftarrow \mathcal{P} \cup \{p_i\}$; $\mathcal{N} \leftarrow \mathcal{N} \cup \{N_i \| \langle 1 \rangle_{\mathsf{E.bl}-\ell}\}$
$\tau_i \leftarrow h_i \oplus p_i$; Return $\tau_i \| C^*_{1,i}$

procedure VF$(1, C_2, H)$ // Games G_0, $\boxed{\mathrm{G}_1}$
$\tau \| C^*_1 \leftarrow C_2$; $h \leftarrow \mathsf{H.Ev}(L, (C^*_1, H))$; $p \leftarrow \tau \oplus h$
If $(p \in \mathcal{P})$ then bad $\leftarrow$ true ; $\boxed{p \leftarrow\!\!\$\ \{0,1\}^{\mathsf{E.bl}} \setminus \mathcal{P}}$
$y \leftarrow \pi^{-1}(p)$; $N \| w \leftarrow y$; $P \leftarrow \mathrm{Pad}^{\pi}_{2,|C^*_1|}(N)$; $M^* \leftarrow C^*_1 \oplus P$; $x \| M \leftarrow M^*$
win $\leftarrow (x = 0^{\ell})$ and $(w = \langle 1 \rangle_{\mathsf{E.bl}-\ell})$; Return false

procedure VF$(1, C_2, H)$ // Games $\mathrm{G}_2, \mathrm{G}_3$
$\tau \| C^*_1 \leftarrow C_2$; $y \leftarrow\!\!\$\ \{0,1\}^{\mathsf{E.bl}} \setminus \mathcal{N}$; $N \| w \leftarrow y$
$P \leftarrow \mathrm{Pad}^{\pi}_{2,|C^*_1|}(N)$; $M^* \leftarrow C^*_1 \oplus P$; $x \| M \leftarrow M^*$
win $\leftarrow (x = 0^{\ell})$ and $(w = \langle 1 \rangle_{\mathsf{E.bl}-\ell})$; Return false

procedure FIN
Return win

Figure 1.19. First set of games used in proof of Theorem 12. Next to procedure names, we indicate the games to which they belong. Unannotated procedures belong to all games in the Figure.

Proof. Consider the games of Figure 1.19. We claim that:

$$\begin{aligned} \mathbf{Adv}^{\mathrm{auth2}}_{\mathsf{SE2}}(A_2) &= \Pr[\mathrm{G}_0(A_2)] && (1.19)\\ &= \Pr[\mathrm{G}_1(A_2)] + (\Pr[\mathrm{G}_0(A_2)] - \Pr[\mathrm{G}_1(A_2)]) \\ &\leq \Pr[\mathrm{G}_1(A_2)] + \Pr[\mathrm{G}_1(A_2) \text{ sets bad}] \ . && (1.20) \end{aligned}$$

Let us now explain games $\mathrm{G}_0, \mathrm{G}_1$ and justify the above. Adversary A_2, by assumption, makes a single query to its NEW oracle, initializing the single user under consideration. Our games pick, for this user, a key K for E, and let L be the corresponding key for H. The adversary

then makes q_e queries to ENC. Since all are directed at user 1, we hardwire 1 as the first input to the oracle, and can think of the adversary queries as triples $(N_1, M_1, H_1), , \ldots, (N_{q_e}, M_{q_e}, H_{q_e})$. The games compute replies correctly according to the encryption algorithm of the scheme. Its ENC queries completed, the adversary makes its single DEC query, which we view as a pair (C_2, H), hardwiring the user number 1 in the oracle. What is returned to the adversary as response does not matter, since the only further action of the adversary is its mandated call to FIN(), and accordingly all our games return false in reply to the DEC query. But internally the games set the win flag, and its value is what FIN() returns as the game output. We assume the adversary's DEC query is non-trivial, meaning $(\tau \| C_1^*, H) \notin \{ (\tau_i \| C_{1,i}^*, H_i) : 1 \le i \le q_e \}$. Game G_0 excludes the boxed code, and thus sets win correctly, justifying Equation (1.19). We will get to the meaning of the boxed code later; for now what matters is that, games $\mathrm{G}_0, \mathrm{G}_1$ being identical-until-bad, the Fundamental Lemma of Game Playing [30] justifies Equation (1.20). This leaves us with two tasks: (1) to bound $\Pr[\mathrm{G}_1(A_2)]$ and (2) to bound $\Pr[\mathrm{G}_1(A_2)$ sets bad$]$.

We start with (1). Game G_2 changes only procedure VF, which, rather than setting $y \leftarrow \pi^{-1}(p)$, picks y at random from $\{0,1\}^{\mathsf{E.bl}} \setminus \mathcal{N}$. We claim this does not change the probability of winning, meaning

$$\Pr[\mathrm{G}_1(A_2)] = \Pr[\mathrm{G}_2(A_2)] \; . \tag{1.21}$$

The justification of Equation (1.21) is that in game G_1, the point p is chosen uniformly at random from $\{0,1\}^{\mathsf{E.bl}} \setminus S$, and π is a permutation, so $y \leftarrow \pi^{-1}(p)$ is distributed uniformly at random in $\{0,1\}^{\mathsf{E.bl}} \setminus \mathcal{N}$. Note that this claim does not rely on any security property of, or security assumption about, the blockcipher E, but only on the fact that $\pi = \mathsf{E.Ev}(K, \cdot)$ is a permutation, which can be regarded as fixed in this argument.

Game G_3 switches π from $\mathsf{E.Ev}(K, \cdot)$ to a random function, the change being in procedure

NEW alone, and we have

$$\Pr[\mathrm{G}_2(A_2)] = \Pr[\mathrm{G}_3(A_2)] + (\Pr[\mathrm{G}_2(A_2)] - \Pr[\mathrm{G}_3(A_2)]) \ .$$

It is now easy to build a prf-adversary B_0 such that

$$\Pr[\mathrm{G}_2(A_2)] - \Pr[\mathrm{G}_3(A_2)] \leq \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{E}}(B_0) \ .$$

The design of B_0 is standard and we omit the details, but we note that the elimination of the computation of π^{-1} was important to be able to rely only on prf security of E, rather than needing to make the stronger assumption that E is prp-cca (also called strong prp) secure.

We are now in a position to exploit the 0^ℓ redundancy that our scheme adds to the message. We claim that

$$\Pr[\mathrm{G}_3] \leq \frac{1}{2^\ell} \ . \tag{1.22}$$

To justify Equation (1.22), we first claim that if game G_3 returns true then $N \notin \{N_1, \ldots, N_{q_e}\}$. If so (we will justify this claim in a bit), π is being invoked on new points (ones to which it has not been already applied in ENC queries) in the computation $P \leftarrow \mathrm{Pad}^{\pi}_{2,|C_1^*|}(N)$, yielding Equation (1.22). Returning to the claim, assume game G_3 returns true. Then it must be that $w = \langle 1 \rangle_{\mathsf{E.bl}-\ell}$. Assume towards a contradiction that $N = N_i$ for some i. Then $y = N \| w = N_i \| \langle 1 \rangle_{\mathsf{E.bl}-\ell}$, putting y in $\mathcal{N}$, but y was drawn from outside $\mathcal{N}$, which is the desired contradiction establishing the claim.

Putting the above together, we have now shown that

$$\Pr[\mathrm{G}_1(A_2)] \leq \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{E}}(B_0) + \frac{1}{2^\ell} \ . \tag{1.23}$$

Games G_4, G_5, G_6

procedure NEW
$K \leftarrow\!\!\$\ \{0,1\}^{\mathsf{E.bl}}$; $\pi \leftarrow \mathsf{E.Ev}(K,\cdot)$ // Games $\mathrm{G}_4, \mathrm{G}_5$
$\pi \leftarrow\!\!\$\ \mathrm{FUNC}(\{0,1\}^{\mathsf{E.bl}}, \{0,1\}^{\mathsf{E.bl}})$ // Game G_6
$L \leftarrow \pi(0^{\mathsf{E.bl}})$

procedure ENC$(1,N,M,H)$
$i \leftarrow i+1$; $N_i \leftarrow N$; $M_i \leftarrow M$; $H_i \leftarrow H$; $P_i \leftarrow \mathrm{Pad}^{\pi}_{2,\ell+|M|}(N)$; $M^* \leftarrow 0^{\ell} \| M$; $C^*_{1,i} \leftarrow M \oplus P_i$
$h_i \leftarrow \mathsf{H.Ev}(L,(C^*_{1,i},H_i))$; $p_i \leftarrow \pi(N_i \| \langle 1 \rangle_{\mathsf{E.bl}-\ell})$; $\mathcal{P} \leftarrow \mathcal{P} \cup \{p_i\}$; $\tau_i \leftarrow h_i \oplus p_i$; Return $\tau_i \| C^*_{1,i}$

procedure VF$(1,C_2,H)$
$\tau \| C^*_1 \leftarrow C_2$; $h \leftarrow \mathsf{H.Ev}(L,(C^*_1,H))$; $p \leftarrow \tau \oplus h$; Return false

procedure FIN // Game G_4
Return $(p \in \mathcal{P})$

procedure FIN // Games $\mathrm{G}_5, \mathrm{G}_6$
Return $(\exists i : ((h \oplus h_i = \tau \oplus \tau_i) \text{ and } (C^*_1,H) \neq (C^*_{1,i},H_i)))$

Game G_7

procedure NEW
$L \leftarrow\!\!\$\ \{0,1\}^{\mathsf{E.bl}}$

procedure ENC$(1,N,M,H)$
$i \leftarrow i+1$; $H_i \leftarrow H$; $C^*_{1,i} \leftarrow\!\!\$\ \{0,1\}^{\ell+|M|}$
$\tau_i \leftarrow\!\!\$\ \{0,1\}^{\mathsf{E.bl}}$; Return $\tau_i \| C^*_{1,i}$

procedure VF$(1,C_2,H)$
$\tau \| C^*_1 \leftarrow C_2$; $h \leftarrow \mathsf{H.Ev}(L,(C^*_1,H))$; Return false

procedure FIN
For $j = 1,\ldots,i$ do $h_i \leftarrow \mathsf{H.Ev}(L,(C^*_{1,i},H_i))$
Return $(\exists i : ((h \oplus h_i = \tau \oplus \tau_i) \text{ and } (C^*_1,H) \neq (C^*_{1,i},H_i)))$

Adversary $C^{\mathrm{INIT,FIN}}$

INIT
$A_2^{\mathrm{NEW}^*,\mathrm{ENC}^*,\mathrm{VF}^*,\mathrm{FIN}^*}$

procedure NEW*
Return

procedure ENC$^*(1,N,M,H)$
$i \leftarrow i+1$; $H_i \leftarrow H$
$C^*_{1,i} \leftarrow\!\!\$\ \{0,1\}^{\ell+|M|}$; $\tau_i \leftarrow\!\!\$\ \{0,1\}^{\mathsf{E.bl}}$
Return $\tau_i \| C^*_{1,i}$

procedure VF$^*(1,C_2,H)$
$\tau \| C^*_1 \leftarrow C_2$; Return false

procedure FIN*
$j \leftarrow\!\!\$\ \{1,\ldots,q_e\}$
FIN$((C^*_1,H),(C^*_{1,j},H_j))$

Figure 1.20. On the top are further games used in the proof of Theorem 12. Lines may be annotated with the names of games which include them, procedures whose names are unannotated belonging to all games. On the bottom left is a final game and on the bottom right is the axu-adversary.

Next we give adversaries B_1, C such that

$$\Pr[\mathrm{G}_1(A_2) \text{ sets } \mathsf{bad}] \leq \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{E}}(B_1) + q_e \cdot \mathbf{Adv}^{\mathrm{axu}}_{\mathsf{H}}(C) \; . \tag{1.24}$$

For this, consider the games of Figure 1.20. We claim

$$\Pr[\mathrm{G}_1(A_2) \text{ sets } \mathsf{bad}] = \Pr[\mathrm{G}_4(A_2)] \tag{1.25}$$

$$= \Pr[\mathrm{G}_5(A_2)] \; . \tag{1.26}$$

Game G_4 results from moving the condition setting bad in G_3 to $\textsc{Fin}()$ and dropping unused code, justifying Equation (1.25). To justify Equation (1.26), we show that if $p \notin \mathcal{P}$ then there exists i such that $(C_1^*, H) \neq (C_{1,i}^*, H_i)$ but $h \oplus \tau = h_i \oplus \tau_i$, meaning there is a (non-trivial) xor computed for $\mathsf{H.Ev}(L, \cdot)$. That $p \in \mathcal{P}$ means there is some i such that $p = p_i$. (This i need not be unique.) So $h \oplus \tau = h_i \oplus \tau_i$. Now assume towards a contradiction that $(C_1^*, H) = (C_{1,i}^*, H_i)$. Since $h = \mathsf{H.Ev}(L, (C_1^*, H))$ and $h_i = \mathsf{H.Ev}(L, (C_{1,i}^*, H_i))$, we get $h = h_i$. But we already had $h \oplus \tau = h_i \oplus \tau_i$, so we have $\tau = \tau_i$. This means $(\tau \| C_1^*, H) = (\tau_i \| C_{1,i}^*, H_i)$, which contradicts the assumption that the $\textsc{Dec}$ query of the adversary is non-trivial. This concludes the justification of Equation (1.26).

Game G_6 switches π from $\mathsf{E.Ev}(K, \cdot)$ to a random function, the change being only in $\textsc{New}$, and we have

$$\Pr[\mathrm{G}_5(A_2)] = \Pr[\mathrm{G}_6(A_2)] + (\Pr[\mathrm{G}_5(A_2)] - \Pr[\mathrm{G}_6(A_2)]) \; .$$

Now we can design adversary B_1 such that

$$\Pr[\mathrm{G}_5(A_2)] - \Pr[\mathrm{G}_6(A_2)] \leq \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{E}}(B_1) \; . \tag{1.27}$$

The design of B_1 is standard and omitted. With π a random function in G_6, the hash key L, and

the ciphertexts returned in G_6 in response to ENC queries, are random, so game G_7 directly picks them that way. This allows it to delay computing the hashes to FIN(). We have

$$\Pr[G_6(A_2)] = \Pr[G_7(A_2)] \ .$$

The bottom right of Figure 1.20 shows our axu-adversary C. It runs A_2, responding to ENC queries with random strings, as per game G_7. It returns, as its two messages, the hash-input for the VF query, and a random one of the q_e hash-inputs for the ENC queries. We have

$$\mathbf{Adv}^{\mathrm{axu}}_{\mathsf{H}}(C) \geq \frac{1}{q_e} \cdot \Pr[G_7(A_2)] \ . \tag{1.28}$$

Putting the above together we have Equation (1.24).

At this point we have shown

$$\mathbf{Adv}^{\mathrm{auth2}}_{\mathsf{SE2}}(A_2) \leq \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{E}}(B_0) + \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{E}}(B_1) + q_e \cdot \mathbf{Adv}^{\mathrm{axu}}_{\mathsf{H}}(C) + \frac{1}{2^{\ell}} \ . \tag{1.29}$$

We merge B_0, B_1 into a single adversary B as follows. Let B pick $c \xleftarrow{\$} \{0,1\}$ and run B_c. Then

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{E}}(B) = \frac{1}{2} \cdot \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{E}}(B_0) + \frac{1}{2} \cdot \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{E}}(B_1) \ . \tag{1.30}$$

Putting together Equations (1.29) and (1.30) concludes the proof. □

A bound on the auth2-advantage of an adversary that makes multiple NEW and VF queries can be obtained, as noted above, by combining our general results with Theorem 12. An interesting open question is to directly analyze such an adversary and obtain a bound better than ours on its auth2-advantage.

1.9 A real-world perspective

In addition to bridging the gap between theory and usage, our framework allows us to formalize weaknesses of real-world schemes which communicate nonces in the clear.

First, it allows us to formalize an intuitive fact: pathologically chosen nonces cannot be communicated in the clear. It may seem obvious that message or key-dependent nonces violate security but such pathological nonce choices have occurred in the wild. For instance, CakePHP, a web framework, used the key as the nonce [1] when encrypting data. The use of a hash of a message has also been proposed, and subsequently argued as insecure, in an Internet forum [118].

Second, it disallows metadata leakage through the nonce. Implicit nonces with a device specific field, such as those recommended in RFC 5116 [104] enable an adversary to distinguish between different user sessions. Even the "standard" nonce choices are not safe against these adversaries. A counter will allow an adversary distinguish between sessions with high traffic and low traffic, and a randomly chosen nonce can detect devices with poor entropy (RSA public keys were used to a similar end by HDWH [79]).

1.10 Acknowledgements

We thank the anonymous reviewers (of the many conferences to which this paper was submitted before finally being accepted at Crypto 2019) for their feedback and suggestions. Bellare was supported in part by NSF grants CNS-1526801 and CNS-1717640, ERC Project ERCC FP7/615074 and a gift from Microsoft. Ng was supported by DSO National Labs. Tackmann was supported in part by the Swiss National Science Foundation (SNF) via Fellowship No. P2EZP2_155566 and NSF grant CNS-1228890.

This chapter, in full, is a reprint of the material as it appears in Advances in Cryptology – CRYPTO 2019. Bellare, Mihir; Ng, Ruth; Tackmann, Björn, Springer Lecture Notes in Computer Science volume 11692, 2019. The dissertation author was the primary investigator and author of

this paper.

Chapter 2

Improved Structured Encryption for SQL Databases via Hybrid Indexing

2.1 Introduction

SQL applications are often deterred from using cloud storage solutions because they do not wish to grant a third party access to their sensitive data. Yet, in-house solutions often are less convenient than these large-scale ones and are vulnerable to compromise as well. This calls for a cryptographic solution which allows data on the cloud to be end-to-end encrypted so that the server never "sees" the sensitive data. This in turn poses a challenge when the server is called upon to perform SQL operations on the data.

Most current offerings of this technology depend heavily on property-revealing encryption (PRE), making them vulnerable to leakage abuse attacks (LAAs). For example, Always Encrypted either deterministically encrypts columns or stores them with an ordered index [9]. These techniques have been shown to offer little-to-no privacy in certain practical scenarios [110, 73].

A more promising approach is structured encryption (StE) which uses auxiliary encrypted data structures (e.g. encrypted multimaps) to support a subset of SQL queries [52]. This is done by translating the SQL query into tokens which can be passed to the server to query the auxiliary structures. The outputs of this are compiled, decrypted and processed to retrieve the SQL query result. Security is measured by *leakage profiles*, which characterize what information a curious

server can learn. In particular, StE-based constructions leak equal or less than PRE-based constructions and resist most known LAAs [59, 110, 35, 37, 71, 73, 76].

Our contributions.

Our work can be grouped into three main contributions:

1. **Partially precomputed joins:** We introduce a new way to index (equi)joins which stems from the simple observation that when the server fully precomputes (FP) joins, the client has to download and decrypt a quadratic number of rows and the server learns the equality pattern of said rows. In our approach, the server partially precomputes (PP) joins: instead of indexing exactly which rows from the input table should be concatenated and returned, it just stores the set of rows from each input table that appears anywhere in the join output. At query time, the client downloads these sets and computes the join. When this is used to support SQL queries of the form "**select** * **from** id_1 **join** id_2 **on** $at_1 = at_2$", PP outperforms FP in both leakage and bandwidth at the cost of a logarithmic factor of client computation (in the worst case).

2. **Hybrid indexing:** When we incorporate PP joins into state-of-the-art StE schemes, we discover that some queries (e.g. those with a selection subquery) cannot be computed in the same way because the server does not know the equality pattern on the join columns (i.e. how the rows "match up"). So while PP joins are still the more secure choice, they sometimes incur more bandwidth than FP. To address this, we develop a hybrid StE scheme with both forms of indexing. The client chooses which to use at query time. We provide the first heuristic (that we are aware of) to enable this type of *leakage-aware client-side query planning*, helping the client decide how to minimize leakage without exceeding a given bandwidth budget.

3. **Simulations on real data:** We quantify the effect of using FP and PP join indexing on bandwidth incurred by simulating our constructions on data from the City of Chicago's

Data Portal and MySQL's sample Sakila database [3, 4]. On simple (non-recursive) join queries, PP's bandwidth is on average 231 times less than FP's but more complex (recursive) queries are split down the middle as to which option used less bandwidth. We also demonstrate the accuracy of our heuristic under different client storage constraints. Assuming client storage comparable to that which is used in SQL Server, our heuristic chose a query plan with the maximal number of PP joins 79% of the time, and the optimal query plan 68% of the time.

Related work.

Encrypted databases have been treated from a variety of perspectives. Structured encryption (StE) was defined by Chase and Kamara (CK) and is a special case of SSE, which was first defined by SWP [128].

We see our work as a direct extension and improvement upon SPX and OPX, two schemes which applied StE to the problem of indexing SQL databases [52, 89, 92]. Both our scheme and OPX address a similar query class to the one introduced in SPX, but lower leakage by using the hashset technique from OXT and primitives inspired by CJJJKRS [48, 49]. In particular, our FpSj scheme in Section 2.4.2 bears many similarities to OPX with minor leakage improvements from using a single indexing data structure. Our PpSj and HybStl schemes (in Section 2.4.2 and Section 2.5 respectively) introduce a new technique which further lowers leakage and server storage. For non-recursive queries, there are also substantial bandwidth savings.

PRE-based solutions achieves higher query support at the cost of higher leakage [116, 70, 2, 62, 134], and are particularly susceptible to leakage abuse attacks [59, 110, 35, 37, 71, 73, 76].

Finally, encrypted search has also been attempted using alternate models and architectures including the database-provider model [77], MPC [53, 20], ORAM [66] and trusted execution environments [95, 45, 16].

Other works have also partially delegated computation to the client, to reduce leakage or increase query support, though none have applied it to joins [130, 54, 57].

2.2 Preliminaries

We denote the empty string with ε. Given positive integer n, let $[n] = \{1, 2, \ldots, n\}$. Given tuples $\mathbf{t}_1 = (x_1, \ldots, x_n)$ and $\mathbf{t}_2 = (y_1, \ldots, y_m)$ we write $\mathbf{t}_1 \| \mathbf{t}_2$ as a shorthand for $(x_1, \ldots, x_n, y_1, \ldots, y_m)$. We extend set operations $\cap, \cup \in, \subseteq$ from sets to tuples by interpreting the tuples as sets.

Our algorithms often make use of dictionaries $\mathbf{D}$ which map labels $\ell \in \{0,1\}^*$ to values $\mathbf{D}[\ell] \in \{0,1\}^* \cup \{\perp\}$. We also adopt the shorthand $\mathbf{D}.\mathsf{Lbls} = \{\ell \in \{0,1\}^* : \mathbf{D}[\ell] \neq \perp\}$. A multimap $\mathbf{M}$ is an dictionary where $\mathbf{M}[\ell]$ is either a set of strings or $\perp$.

Pseudocode.

In pseudocode, we will assume that all integers , strings and sets are initialized to 0 , ε and $\emptyset$ respectively. For dictionaries and multimaps, they are initialized with all labels mapping to $\perp$. If S is a set or dictionary value, we write $S \xleftarrow{\cup} x$ in pseudocode as a shorthand for $S \leftarrow S \cup \{x\}$, initializing it first to $\emptyset$ if necessary. If $\mathbf{t}$ is a tuple, we similarly mean $\mathbf{t} \leftarrow \mathbf{t} \| (x)$ by writing $\mathbf{t} \leftarrow \mathbf{t} \xleftarrow{\cup} x$. Finally, we will write "Define X : *pred*" to set X (a function or constant) in such a way that the predicate *pred* is true. If there are undefined variables in *pred* we treat it as a random variable and expect that X is defined such that *pred* will always be true.

Games.

Our work uses the code-based game-playing framework of BR [30]. Let G be a game and A an adversary. Then, we write $\Pr[\mathrm{G}(A)]$ to denote the probability that A plays G and the latter returns true. G may provide oracles to A, and if so we write $A^{\mathrm{O}_1, \ldots, \mathrm{O}_n}$ to denote that A is run with access to oracles $\mathrm{O}_1, \ldots, \mathrm{O}_n$.

Symmetric Encryption, IND$-security.

Symmetric Encryption (SE) scheme SE defines key set $\mathsf{SE.KS}$, encryption algorithm $\mathsf{SE.Enc}$ and decryption algorithm $\mathsf{SE.Dec}$. Encryption is randomized, taking a key $K_{\mathrm{e}} \in \mathsf{SE.KS}$ and a message $M \in \{0,1\}^*$ and returns a ciphertext $C \in \{0,1\}^*$. Decryption is deterministic and

Game $\mathrm{G}^{\mathrm{ind\$}}_{\mathsf{SE}}(A)$	**Game** $\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(A)$
$b \leftarrow\!\!\$\, \{0,1\}$; $K_{\mathrm{e}} \leftarrow\!\!\$\, \mathsf{SE.KS}$	$b \leftarrow\!\!\$\, \{0,1\}$; $K_{\mathrm{f}} \leftarrow\!\!\$\, \mathsf{F.KS}$
$b' \leftarrow\!\!\$\, A^{\mathrm{ENC}}$; Return $b = b'$	$b' \leftarrow\!\!\$\, A^{\mathrm{FN}}$; Return $b = b'$
Alg $\mathrm{ENC}(m)$	**Alg** $\mathrm{FN}(X)$
$c_1 \leftarrow\!\!\$\, \mathsf{SE.Enc}(K_{\mathrm{e}}, m)$	If $\mathbf{C}[X] = \perp$ then $\mathbf{C}[X] \leftarrow\!\!\$\, \{0,1\}^{\mathsf{F.ol}}$
$c_0 \leftarrow\!\!\$\, \{0,1\}^{\lvert c_1 \rvert}$; Return c_b	$c_1 \leftarrow\!\!\$\, \mathsf{F.Ev}(K_{\mathrm{f}}, X)$; $c_0 \leftarrow \mathbf{C}[X]$; Return c_b

Figure 2.1. Games used in defining IND$ security of SE scheme SE (right) and PRF security of function family F (left)

takes a key and ciphertext, returning a message. SE also defines a ciphertext length function SE.cl. We require that if $C \leftarrow\!\!\$\, \mathsf{SE.Enc}(K_{\mathrm{e}}, M)$ then $|C| = \mathsf{SE.cl}(|M|)$ and $\Pr[\mathsf{SE.Dec}(K_{\mathrm{e}}, C) = M] = 1$. We want our SE schemes to protect the privacy of M, so ciphertexts should be indistinguishable from a random string of length $\mathsf{SE.cl}(|M|)$. We capture this with the game $\mathrm{G}^{\mathrm{ind\$}}_{\mathsf{SE}}$ in Fig. 2.1 and say that a scheme is IND$-secure if $\mathbf{Adv}^{\mathrm{ind\$}}_{\mathsf{SE}}(A) = 2\Pr[\mathrm{G}^{\mathrm{ind\$}}_{\mathsf{SE}}(A)] - 1$ is small for all adversaries A.

Function Families, PRF-security.

A function family F defines a key set F.KS and an output length F.ol. It defines a deterministic evaluation algorithm $\mathsf{F.Ev}\colon \mathsf{F.KS} \times \{0,1\}^* \to \{0,1\}^{\mathsf{F.ol}}$. We define PRF security for function family F via the game $\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}$ depicted in Fig. 2.1. We say that F is a PRF if $\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(A) = 2\Pr[\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(A)] - 1$ is small for all adversaries A.

2.3 Structured Indexing for SQL data types

We now generalize CK's definition of structured encryption and provide a new framework for modeling encrypted SQL systems [52].

Abstract Data Types.

An *abstract data type* ADT defines a domain set ADT.Dom, a query set ADT.QS, and a deterministic specification function $\mathsf{ADT.Spec} : \mathsf{ADT.Dom} \times \mathsf{ADT.QS} \to \{0,1\}^*$.

An example is the dictionary ADT DyAdt. DyAdt.Dom, DyAdt.QS contain all possible

dictionaries $\mathbf{D}$ and labels respectively (as defined in Section 2.2), and $\mathsf{DyAdt.Spec}(\mathbf{D}, \ell) = \mathbf{D}[\ell]$. Multimap ADT MmAdt is defined analogously.

Structured Indexing.

We generalize Structured Encryption (StE) schemes (as defined by CK [52]) to *structured indexing* (StI) schemes. These are StE schemes without a decryption algorithm. The intuition here is that the handling of outsourced data often indexes the data in addition to encrypting it and we would like these encrypted indexes, whatever form they take, to achieve semantic security as well. Later, we show how this primitive allows us to modularize StE schemes. A StI scheme Stl for ADT defines a set of keys $\mathsf{Stl.KS}$ and the following algorithms:

- Randomized encryption algorithm $\mathsf{Stl.Enc}$ which takes a key $K' \in \mathsf{Stl.KS}$ and an element of $\mathsf{ADT.Dom}$ and returns an updated key K and index $\mathsf{IX} \in \{0,1\}^*$. This syntax generalizes that of CK by allowing key generation to occur within or outside $\mathsf{Stl.Enc}$.

- Possibly randomized token generation algorithm $\mathsf{Stl.Tok}$ which takes a key and a query from $\mathsf{ADT.QS}$, and returns fixed length token $\mathsf{tk} \in \{0,1\}^{\mathsf{Stl.tl}}$.

- Deterministic evaluation algorithm $\mathsf{Stl.Eval}$ which takes a token and index, and returns a ciphertext string $C \in \{0,1\}^*$.

- Finalization algorithm $\mathsf{Stl.Fin}$ which takes K, q and an input string, and returns an output string.

Intuitively, the client indexes his data then encrypts this index with $\mathsf{Stl.Enc}$, storing IX on the server. At query time, the client uses $\mathsf{Stl.Tok}$ to generate a token and sends it to the server who runs $\mathsf{Stl.Eval}$, returning C to the client. $\mathsf{Stl.Fin}$ can be used for client-side post-processing of the data. Note that the output of $\mathsf{Stl.Eval}$ need not be the input to $\mathsf{Stl.Fin}$. In our indexing schemes the server will use the output of $\mathsf{Stl.Eval}$ as "pointers" to retrieve rows of SQL data stored in a different data structure which in turn form the input to $\mathsf{Stl.Fin}$.

$\underline{\textbf{Alg } \mathsf{Dye}_\pi.\mathsf{Enc}\big((K_\mathrm{f},K_\mathrm{e}),\mathbf{D}\big)}$
Pad all values in $\mathbf{D}$ to the same length
For $\ell \in \mathbf{D}.\mathsf{Lbls}$ do $\mathbf{D}'[\mathsf{F.Ev}(K_\mathrm{f},\ell)] \leftarrow\!\!\$\ \mathsf{SE.Enc}(K_\mathrm{e},\mathbf{D}[\ell])$
Return $\big((K_\mathrm{f},K_\mathrm{e}),\mathbf{D}'\big)$

$\underline{\textbf{Alg } \mathsf{Dye}_\pi.\mathsf{Tok}\big((K_\mathrm{f},K_\mathrm{e}),\ell\big)}$
$\mathsf{tk} \leftarrow \mathsf{F.Ev}(K_\mathrm{f},\ell)$; Return tk

$\underline{\textbf{Alg } \mathsf{Dye}_\pi.\mathsf{Eval}(\mathsf{tk},\mathbf{D}')}$
Return $\mathbf{D}'[\mathsf{tk}]$

$\underline{\textbf{Alg } \mathsf{Dye}_\pi.\mathsf{Dec}\big((K_\mathrm{f},K_\mathrm{e}),C\big)}$
Unpad and return $\mathsf{SE.Dec}(K_\mathrm{e},C)$

$\underline{\textbf{Alg } \mathsf{Mme}^{\mathrm{rr}}_\pi.\mathsf{Enc}(K_\mathrm{f},\mathbf{M})}$
Pad all values in $\mathbf{M}$ to the same length
For $\ell \in \mathbf{M}.\mathsf{Lbls}$ do
  $K_\mathrm{e} \leftarrow \mathsf{F.Ev}(K_\mathrm{f},\ell\|0)$; $K \leftarrow \mathsf{F.Ev}(K_\mathrm{f},\ell\|1)$
  For $v \in \mathbf{M}[\ell]$ do
    $\mathbf{D}[\mathsf{F.Ev}(K,\mathrm{ctr})] \leftarrow\!\!\$\ \mathsf{SE.Enc}(K_\mathrm{e},v)$; $\mathrm{ctr} \leftarrow \mathrm{ctr}+1$
Return $(K_\mathrm{f},\mathbf{D})$

$\underline{\textbf{Alg } \mathsf{Mme}^{\mathrm{rr}}_\pi.\mathsf{Tok}(K_\mathrm{f},\ell)}$
Return $\big(\mathsf{F.Ev}(K_\mathrm{f},\ell\|0),\mathsf{F.Ev}(K_\mathrm{f},\ell\|1)\big)$

$\underline{\textbf{Alg } \mathsf{Mme}^{\mathrm{rr}}_\pi.\mathsf{Eval}\big((K_\mathrm{e},K),\mathbf{D}\big)}$
While $\mathbf{D}[\mathsf{F.Ev}(K,\mathrm{ctr})] \neq \perp$ do
  $x \leftarrow \mathsf{SE.Dec}\big(K_\mathrm{e},\mathbf{D}[\mathsf{F.Ev}(K,\mathrm{ctr})]\big)$
  $\mathrm{ctr} \leftarrow \mathrm{ctr}+1$; Unpad x then $S \xleftarrow{\cup} x$
Return S

Figure 2.2. Algorithms for RH dictionary encryption scheme Dye_π and RR multimap encryption scheme $\mathsf{Mme}^{\mathrm{rr}}_\pi$.

Structured Encryption.

We can now define StE as a special cases of StI. Intuitively, an StE scheme is an StI scheme where the data structure is also used to store query responses (as opposed to just indexing them). The output of evaluation can be fed into finalization for decryption and should return the query result. To highlight this, StE schemes have a *decryption algorithm* $\mathsf{StE.Dec}$ in place of a finalization algorithm which takes as input K,q,C and returns the query result. We define correctness via game $\mathrm{G}^{\mathrm{cor}}_{\mathsf{StE}}$ in Fig. 2.3 and say that StE is correct if the advantage of all adversaries A, defined $\mathbf{Adv}^{\mathrm{cor}}_{\mathsf{StE}}(A) = \Pr[\mathrm{G}^{\mathrm{cor}}_{\mathsf{StE}}(A)]$, is low. The correctness of our schemes will depend on the collision resistance of their function family primitives. Since we assume these are PRFs to prove security, we will also assume that their key-lengths are sufficient to ensure correctness.

We subdivide StE schemes into two types. We say that a scheme $\mathsf{StE}_{\mathrm{rr}}$ is *response revealing* (RR) if evaluation itself returns the query result. In other words, decryption must be such that $\mathsf{StE}_{\mathrm{rr}}.\mathsf{Dec}(K,q,C) = C$ for all K,q,C. An StE scheme that is not RR is *response hiding* (RH).

Game $\mathrm{G}^{\mathrm{cor}}_{\mathsf{StE}}(A)$	**Game** $\mathrm{G}^{\mathrm{ss}}_{\mathsf{StI},\mathcal{L},\mathcal{S}}(A)$
$(\mathsf{DS}, st) \leftarrow\!\!\$\, A(\mathtt{s})$; $K' \leftarrow\!\!\$\, \mathsf{StE.KS}$	$(\mathsf{DS}, (q_1, \ldots, q_n), st) \leftarrow\!\!\$\, A(\mathtt{s})$
If $\mathsf{DS} \notin \mathsf{ADT.Dom}$ then return false	$b \leftarrow\!\!\$\, \{0,1\}$; $K' \leftarrow\!\!\$\, \mathsf{StI.KS}$
$(K, \mathsf{EDS}) \leftarrow\!\!\$\, \mathsf{StE.Enc}(K', \mathsf{DS})$	If $\mathsf{DS} \notin \mathsf{ADT.Dom}$ or $\{q_i\}_{i=1}^n \not\subseteq \mathsf{ADT.QS}$ then
$A^{\textsc{Tok}}(\mathtt{g}, \mathsf{EDS}, st)$; Return win	Return false
Oracle $\textsc{Tok}(q)$	If $b = 1$ then
If $q \notin \mathsf{ADT.QS}$ then win $\leftarrow$ false	$(K, \mathsf{IX}) \leftarrow\!\!\$\, \mathsf{StI.Enc}(K', \mathsf{DS})$
$C \leftarrow \mathsf{StE.Eval}(\mathsf{StE.Tok}(K, q), \mathsf{EDS})$	For $i \in [n]$ do $\mathsf{tk}_i \leftarrow\!\!\$\, \mathsf{StI.Tok}(K, q_i)$
$M \leftarrow \mathsf{StE.Dec}(K, C)$	Else
If $\mathsf{ADT.Spec}(\mathsf{DS}, q) \neq M$ then win $\leftarrow$ false	$(\mathsf{IX}, (\mathsf{tk}_1, \ldots, \mathsf{tk}_n)) \leftarrow\!\!\$\, \mathcal{S}(\mathcal{L}(\mathsf{DS}, (q_1, \ldots, q_n)))$
Return tk	$b' \leftarrow\!\!\$\, A(\mathtt{g}, \mathsf{IX}, (\mathsf{tk}_1, \ldots, \mathsf{tk}_n), st)$; Return $(b = b')$

Figure 2.3. Games used in defining correctness for StE (structured encryption scheme for ADT) and semantic security for StI (structured indexing scheme for ADT) with respect to leakage algorithm $\mathcal{L}$ and simulator $\mathcal{S}$.

Dictionary/Multimap Encryption.

We refer to StE for the multimap and dictionary data types as multimap and dictionary encryption (MME/DYE) respectively.

Our constructions make use of a specific dictionary encryption scheme Dye_π adapted from CJJ+'s SSE scheme $\prod_{\mathrm{bas}}$ (2Lev in the Clusion library) [48, 101]. In this scheme, the encrypted data structure is itself a dictionary $\mathbf{D}'$. We start by padding all values in the input dictionary to the same length, then for each label-value pair $\ell, \mathbf{D}[\ell]$, we do $\mathbf{D}'[\mathsf{F.Ev}(K_{\mathrm{f}}, \ell)] \leftarrow \mathsf{SE.Enc}(K_{\mathrm{e}}, \mathbf{D}[\ell])$ where F is a pseudorandom function family and SE is a symmetric encryption scheme. The pseudocode for Dye_π is given in Fig. 2.2. The primitives (given as input to **SqlStE**) used in Dye_π are symmetric encryption scheme SE and function family F. Note that $\mathsf{Dye}_\pi.\mathsf{KS} = \mathsf{F.KS} \times \mathsf{SE.KS}$.

Our constructions also make use of a generic RR multimap encryption scheme. We adapt Dye_π to $\mathsf{Mme}^{\mathrm{rr}}_\pi$ (using a counter and label-dependent K_{e}) as an example of such a scheme. Its algorithms are also in Fig. 2.2. The primitives are as in Dye_π but we require that $\mathsf{SE.KS} = \{0,1\}^{\mathsf{F.ol}}$. Note that $\mathsf{Mme}^{\mathrm{rr}}_\pi.\mathsf{KS} = \mathsf{F.KS}$.

Semantic security.

We define semantic security for StI using game $\mathrm{G}^{\mathrm{ss}}_{\mathsf{StI},\mathcal{L},\mathcal{S}}$ depicted in Fig. 2.3, where StI is

a StI scheme for ADT and $\mathcal{L}, \mathcal{S}$ are algorithms we refer to as the leakage algorithm and simulator respectively. The adversary runs in a setup and guessing phase, as indicated by the first argument to it. Its advantage is $\mathbf{Adv}^{\text{ss}}_{\mathsf{StI},\mathcal{L},\mathcal{S}}(A) = 2\Pr[\mathrm{G}^{\text{ss}}_{\mathsf{StI},\mathcal{L},\mathcal{S}}(A) = 1] - 1$. Note that when StI is an StE scheme we recover CK's non-adaptive security notion.

2.3.1 SQL Data Types

We now describe our notation for SQL data, queries and operations. We then define a class of ADTs we call *SQL data types* to construct StE schemes for.

SQL relations, databases, schemas.

SQL relation R defines a tuple of distinct attributes $\mathsf{R.Ats} = (at_1, \ldots, at_n)$. Each attribute is a bitstring $at \in \{0,1\}^*$ and represents a "column" in the relation. R also defines a table R.T consisting of n-tuples of bitstrings representing the "rows" in the relation. Given a row $(x_1, \ldots, x_n) = \mathbf{r} \in \mathsf{R.T}$, we refer to the i-th entry of the row with $\mathbf{r}[at_i] = x_i$. We can initialize a relation with $\mathsf{NewRltn}(\mathbf{at})$ which returns the relation with $\mathsf{R.Ats} = \mathbf{at}$ and no rows.

We define a *database* to be a set of relations with disjoint attributes and their (distinct) identifiers, i.e. a set of the form $\mathsf{DB} = \{(id_1, \mathsf{R}_1), \ldots, (id_N, \mathsf{R}_N)\}$ where $i \neq j$ implies $id_i \neq id_j$ and $\mathsf{R}_i.\mathsf{Ats} \cap \mathsf{R}_j.\mathsf{Ats} = \emptyset$. We denote the identifier set of such a database as $\mathsf{DB.IDs} = \{id_i\}_{i \in [N]}$ and retrieve relations by identifier using $\mathsf{DB}[id_i] = \mathsf{R}_i$. Since database attributes are non-repeating, we allow the retrieval of a table by any of its attributes using getID (i.e. if $\mathsf{getID}(at, \mathsf{DB}) = id$ then $at \in \mathsf{DB}[id].\mathsf{Ats}$). Similarly, if $\mathbf{t} \subseteq \mathsf{DB}[id].\mathsf{Ats}$, then $\mathsf{getID}(\mathbf{t}, \mathsf{DB}) = id$.

We require that each $(id, \mathsf{R}) \in \mathsf{DB}$ has a *unique key attribute* $\mathsf{uk}(id) \in \mathsf{R.Ats}$. This functions as a "row number" which uniquely identifies each row. In other words, for all distinct $\mathbf{r}, \mathbf{r}' \in \mathsf{R.T}$, we have $\mathbf{r}[\mathsf{uk}(id)] \neq \mathbf{r}[\mathsf{uk}(id')]$. Given some $\mathbf{r} \in \mathsf{DB}[id]$ we refer to the tuple $(id, \mathbf{r}[\mathsf{uk}(id)])$ as its *coordinates* and note that it uniquely identifies that row within the database. Additionally, we refer to the values in a "column" with $\mathsf{rng}(at, \mathsf{DB}) = \{\mathbf{r}[at] : \mathbf{r} \in \mathsf{DB}[\mathsf{getID}(at, \mathsf{DB})]\}$.

A database's schema communicates all information about DB except the tables: $\mathsf{Schema}(\mathsf{DB}) = \{(id, \mathsf{R.Ats}) : (id, \mathsf{R}) \in \mathsf{DB}\}$. As shorthand, if $\mathsf{scma} = \mathsf{Schema}(\mathsf{DB})$ then $\mathsf{scma}[id] = \mathsf{DB}[id].\mathsf{Ats}$ and $\mathsf{getID}(at, \mathsf{scma}) = \mathsf{getID}(at, \mathsf{DB})$. In our schemes, the client stores $\mathsf{Schema}(\mathsf{DB})$ as part of the key in order to appropriately format data returned by the server. This is a result of our explicit handling of schemas, coordinates and attributes, something which was left implicit in prior work.

SQL operations.

In our work, we address the secure computation of SQL (equi)joins and (equality) selections. These operations work as follows.

The selection operation is parametrized by a pair of bitstrings (at, x), takes a relation R_1 with $at \in \mathsf{R.Ats}$ as input, and returns $\mathsf{R} = \sigma_{(at,x)}(\mathsf{R}_1)$ where:

$$\mathsf{R.Ats} = \mathsf{R}_1.\mathsf{Ats} \text{ and } \mathsf{R.T} = \{\mathbf{r} \in \mathsf{R}_1.\mathsf{T} : \mathbf{r}[at] = x\}.$$

In Fig. 2.4, we provide an example of such a selection on a relation in a database.

The join infix operation is a function parametrized by two equal-length tuples of attributes $\mathbf{t}_1, \mathbf{t}_2$. It takes two relations $\mathsf{R}_1, \mathsf{R}_2$ with disjoint attribute sets where $(at_1^i, \ldots, at_n^i) = \mathbf{t}_i \subseteq \mathsf{R}_i.\mathsf{Ats}$. It returns $\mathsf{R} = \mathsf{R}_1 \bowtie_{\mathbf{t}_1,\mathbf{t}_2} \mathsf{R}_2$ where:

$$\mathsf{R.Ats} = \mathsf{R}_1.\mathsf{Ats} \| \mathsf{R}_2.\mathsf{Ats} \qquad \text{and}$$
$$\mathsf{R.T} = \{\mathbf{r}_1 \| \mathbf{r}_2 : \mathbf{r}_1 \in \mathsf{R}_1.\mathsf{T},\ \mathbf{r}_2 \in \mathsf{R}_2.\mathsf{T}, \forall i \in [n], \mathbf{r}_1[at_i^1] = \mathbf{r}_2[at_i^2]\}.$$

In the case of a join on singleton tuples, we abbreviate $\bowtie_{(at),(at')}$ as $\bowtie_{at,at'}$. In Fig. 2.4, we provide an example such a join. Attribute tuples can be empty in which case it returns the Cartesian product of the input rows. This is also known as the "cross" operation $\times$.

$\mathsf{R}_1.\mathsf{T}$

$\mathsf{uk}(id_1)$	at_1
aa	Alice
bb	Alice
cc	Bob
dd	Charlie
ee	David

$\mathsf{R}_2.\mathsf{T}$

$\mathsf{uk}(id_2)$	at_2	at_3
11	Alice	Math
22	Alice	Chem
33	Bob	CS
44	Eve	CS
55	Eve	Bio

$(\sigma_{at_2,\mathtt{Eve}}(\mathsf{R}_2)).\mathsf{T}$

$\mathsf{uk}(id_2)$	at_2	at_3
44	Eve	CS
55	Eve	Bio

$(\sigma_{at_3,\mathtt{CS}}(\mathsf{R}_1 \bowtie_{at_1,at_2} \mathsf{R}_2)).\mathsf{T}$

$\mathsf{uk}(id_1)$	at_1	$\mathsf{uk}(id_2)$	at_2	at_3
cc	Bob	33	Bob	CS

$(\mathsf{R}_1 \bowtie_{at_1,at_2} \mathsf{R}_2).\mathsf{T}$

$\mathsf{uk}(id_1)$	at_1	$\mathsf{uk}(id_2)$	at_2	at_3
aa	Alice	11	Alice	Math
aa	Alice	22	Alice	Chem
bb	Alice	11	Alice	Math
bb	Alice	22	Alice	Chem
cc	Bob	33	Bob	CS

Figure 2.4. Examples of SQL relations $\mathsf{R}_1, \mathsf{R}_2$ and the output of join ($\bowtie$) and select (σ) operations on them.

ADT for SQL databases.

We say that an ADT SqlDT is a *SQL data type* if its domain elements $\mathsf{DB} \in \mathsf{SqlDT}$ are *SQL databases* which take the form $\mathbf{DB} = (\mathsf{DB}, \alpha)$ where DB is as defined in Section 2.3.1 and $\alpha \in \{0,1\}^*$ is the auxiliary data. The purpose of α is to allow annotations on DB consistent with real world applications. In this work, we use α to indicate the allowed joins, and $\mathsf{SqlDT.Spec}$ always returns either a relation or $\perp$.

2.3.2 Constructing StE for SQL Data Types Using Encrypted Indexes

Our end goal is structurally encrypted databases supporting response-hiding SQL queries. We build these by constructing StI schemes for classes of SQL queries, then converting these into StE schemes for SQL data types via a generic transform. We now describe this conversion, then dedicate the remainder of this work to the abovementioned StI schemes.

StE, StI for SqlDT.

Intuitively, our StE schemes handle the indexing and storage of SQL data separately. We do the former with an StI scheme and the latter with an RH dictionary encryption scheme. This modularization simplifies pseudocode and reduces the problem of designing secure StE schemes to that of StI schemes.

More formally, we construct an StE scheme for SQL data type SqlDT using the transform **SqlStE** which which takes uses an StI scheme for SqlDT_1 (described below), symmetric encryption scheme SE and function family F. We capture the syntax and pseudocode of StE's algorithms in Fig. 2.5. Note that $\mathsf{StE.KS} = \mathsf{StI.KS}$ and Dye_π is the RH dictionary encryption scheme given in Section 2.3 which uses SE, F as primitives. It is used in $\mathsf{EncRows}, \mathsf{EvalRows}, \mathsf{DecRows}$, which encrypt, retrieve and decrypt the rows of database DB. We used a specific RH dictionary encryption scheme because pathological alternatives may introduce circular security issues, preventing a more general approach.

We now describe how the algorithms in StI and $\mathsf{StE} = \mathbf{SqlStE}[\mathsf{StI}, \mathsf{SE}, \mathsf{F}]$ work. During

Alg $\mathsf{StE.Enc}(K'_{\mathrm{i}}, \mathbf{DB})$

$(K_{\mathrm{d}}, ED, \mathsf{DS}) \leftarrow\!\!\$\ \mathsf{EncRows}(\mathbf{DB})$; $(K_{\mathrm{i}}, \mathsf{IX}) \leftarrow\!\!\$\ \mathsf{StI.Enc}(K'_{\mathrm{i}}, \mathsf{DS})$; Return $\big((K_{\mathrm{d}}, K_{\mathrm{i}}), (ED, \mathsf{IX})\big)$

Subroutine $\mathsf{EncRows}\big((\mathsf{DB}, \alpha)\big)$

For $(id, \mathsf{R}) \in \mathsf{DB}$ do
  For $\mathbf{r} \in \mathsf{R.T}$ do $\mathbf{D}[(id, \mathbf{r}[\mathsf{uk}(id)])] \leftarrow \mathbf{r}$
$(K_{\mathrm{d}}, ED) \leftarrow\!\!\$\ \mathsf{Dye}_{\pi}.\mathsf{Enc}(\mathbf{D})$
For $(id, \mathsf{R}) \in \mathsf{DB}$ do
  For $\mathbf{r} \in \mathsf{R.T}$ do $\ell \leftarrow (id, \mathbf{r}[\mathsf{uk}(id)])$; $\mathbf{T}[\ell] \leftarrow \mathsf{Dye}_{\pi}.\mathsf{Tok}\big(K_{\mathrm{d}}, \ell\big)$
Return $\big(K_{\mathrm{d}}, ED, (\mathsf{DB}, \alpha, \mathbf{T})\big)$

Alg $\mathsf{StE.Tok}\big((K_{\mathrm{d}}, K_{\mathrm{i}}), q\big)$

$\mathsf{tk} \leftarrow\!\!\$\ \mathsf{StI.Tok}(K_{\mathrm{i}}, q)$; Return tk

Alg $\mathsf{StE.Eval}\big(\mathsf{tk}, (ED, \mathsf{IX})\big)$

$P \leftarrow \mathsf{StI.Eval}(\mathsf{tk}, \mathsf{IX})$; Return $\mathsf{EvalRows}(P, ED)$

Subroutine $\mathsf{EvalRows}\big((P_1, \ldots, P_n), ED\big)$

For $i \in [n]$ do $C_i \leftarrow \{(c_1, \ldots, c_{n'}) : (\mathsf{rt}_1, \ldots, \mathsf{rt}_{n'}) \in P_i\ ,\ c_j = \mathsf{Dye}_{\pi}.\mathsf{Eval}(\mathsf{rt}_j, ED)\}$
$C \leftarrow (C_1, \ldots, C_n)$; Return C

Alg $\mathsf{StE.Dec}\big((K_{\mathrm{d}}, K_{\mathrm{i}}), q, C\big)$

Return $\mathsf{StI.Fin}(K_{\mathrm{i}}, q, \mathsf{DecRows}(K_{\mathrm{d}}, C))$

Subroutine $\mathsf{DecRows}\big(K_{\mathrm{d}}, (C_1, \ldots, C_n)\big)$

For $i \in [n]$ do $M_i \leftarrow \{(m_1, \ldots, m_{n'}) : (c_1, \ldots, c_{n'}) \in P_i\ ,\ m_j = \mathsf{Dye}_{\pi}.\mathsf{Eval}(\mathsf{rt}_j, ED)\}$
$M \leftarrow (M_1, \ldots, M_n)$; Return M

Alg $\mathcal{L}\big(\mathbf{DB}, (q_1, \ldots, q_n)\big)$

$(K, ED, \mathsf{DS}) \leftarrow\!\!\$\ \mathsf{EncRows}(\mathbf{DB})$ using a random function in place of $\mathsf{F.Ev}(K_{\mathrm{f}}, \cdot)$
Let L, N be the max row length and # of rows in **DB**
$lk^{\mathrm{i}} \leftarrow\!\!\$\ \mathcal{L}^{\mathrm{i}}(\mathsf{DS}, (q_1, \ldots, q_n))$; Return (lk^{i}, N, L)

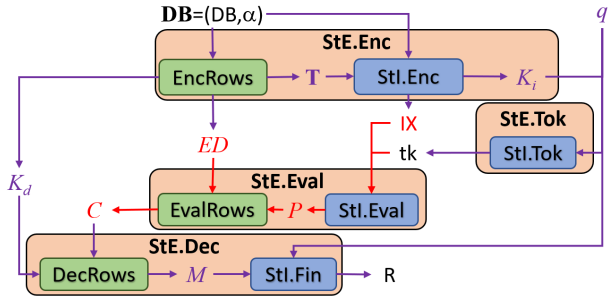

Figure 2.5. Algorithms and for structured encryption scheme $\mathsf{StE} = \mathbf{SqlStE}[\mathsf{StI}, \mathsf{SE}, \mathsf{F}]$ expressed both in pseudocode (top) and diagrammatically (bottom), and its leakage algorithm $\mathcal{L}$ (middle). Dye_{π} is the RH dictionary encryption scheme Dye_{π} in Section 2.3 and $\mathcal{L}^{\mathrm{i}}$ is StI's leakage profile.

StE.Enc, algorithm EncRows will store the rows of DB in an encrypted dictionary ED using Dye_π.Enc. It also prepares a token dictionary $\mathbf{T}$ which maps each row coordinate to a token for Dye_π. SQL data type SqlDT_1 is the same as SqlDT except that its domain elements now take the form $\mathsf{DS} = (\mathsf{DB}, \alpha, \mathbf{T})$ where $(\mathsf{DB}, \alpha) \in \mathsf{SqlDT.Dom}$. The output of StE.Enc is ED and the index returned by StI.Enc(DS).

StE's tokens are those generated by StI. As such, the server's first step in StE.Eval is to run StI.Eval. We require that StI.Eval returns a *pointer tuple* $P = (P_1, \ldots, P_n)$ which is a tuple of sets of tokens. The tokens in each P_i come from $\mathbf{T}$ and point to rows from the same table. Algorithm EvalRows replaces each token with relevant (encrypted) row from ED and returns *ciphertext tuple* $C = (C_1, \ldots, C_n)$, the output of StE.Eval.

During StE.Dec, algorithm DecRows decrypts each ciphertext to get *plaintext tuple* $M = (M_1, \ldots, M_n)$. StI.Fin takes these decrypted rows and performs any final client-side post-processing, returning the final output relation R.

In this work, we will define three different SQL data types, each with its own StI scheme(s). To demonstrate that all of these can be used to construct secure RH StE for their respective data type via **SqlStE**, we demonstrate that the semantic security of StE reduces to that of its primitives. The proof follows a standard hybrid argument.

Theorem 13 *Let* $\mathsf{StE} = \mathbf{SqlStE}[\mathsf{StI}, \mathsf{SE}, \mathsf{F}]$ *be a correct StE scheme for* SqlDT. *Then given algorithms* $\mathcal{L}^{\mathrm{i}}, \mathcal{S}^{\mathrm{i}}$ *and adversary A we can define* $\mathcal{L}$ *as in Fig. 2.5 and construct* $\mathcal{S}, A_{\mathrm{s}}, A_{\mathrm{f}}, A_{\mathrm{i}}$ *such that:*

$$\mathbf{Adv}^{\mathrm{ss}}_{\mathsf{StE},\mathcal{L},\mathcal{S}}(A) \leq \mathbf{Adv}^{\mathrm{ind\$}}_{\mathsf{SE}}(A_{\mathrm{s}}) + \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(A_{\mathrm{f}}) + \mathbf{Adv}^{\mathrm{ss}}_{\mathsf{StI},\mathcal{L}^{\mathrm{i}},\mathcal{S}^{\mathrm{i}}}(A_{\mathrm{i}}).$$

Proof. The adversaries, simulator and games $\mathrm{G}_0, \mathrm{G}_1, \mathrm{G}_2, \mathrm{G}_3$ are given in Fig. 2.6. Notice that the EncRows algorithm used in the adversaries and games is given at the top, and uses two oracles ENC, FN which the algorithms define. Let b be the challenge bit selected in $\mathrm{G}^{\mathrm{ss}}_{\mathsf{StE},\mathcal{L},\mathcal{S}}(A)$.

Notice that we can express $\mathbf{Adv}^{\mathrm{ss}}_{\mathsf{StE},\mathcal{L},\mathcal{S}}(A) = \Pr[\mathrm{G}^{\mathrm{ss}}_{\mathsf{StE},\mathcal{L},\mathcal{S}}(A) | b = 1] - \Pr[\mathrm{G}^{\mathrm{ss}}_{\mathsf{StE},\mathcal{L},\mathcal{S}}(A) | b =$

$0] = \Pr[G_3] - \Pr[G_0]$. In $b = 1$ case, this follows directly from the definition of A_i. In the $b = 0$ case, this follows from the definition of $\mathcal{L}^i, \mathcal{S}^i$.

The only difference between G_0 and G_1 is whether $\mathsf{IX}, \mathsf{tk}_1, \ldots, \mathsf{tk}_n$ are generated using Stl's algorithms or $\mathcal{S}$. In both cases, $\mathbf{D}'$'s values are encrypted using SE.Enc. This is the same differentiation going on in the semantic security game so $G^{ss}_{\mathsf{Stl},\mathcal{L}^i,\mathcal{S}^i}(A_i) = \Pr[G_1] - \Pr[G_0]$. Similarly the difference between G_1 and G_2 is whether the values in $\mathbf{D}'$ are the output of SE.Enc or random strings which is what is going on in the IND$-security game $G^{ind\$}_{\mathsf{SE}}(A_s)$, so $\mathbf{Adv}^{ind\$}_{\mathsf{SE}}(A_s) = \Pr[G_2] - \Pr[G_1]$. Once again, the difference between G_2 and G_3 is whether the labels in $\mathbf{D}'$ (i.e. the tokens in Dye_π.Enc) are generated using F.Ev or a random function which is what is going on in the PRF-security game $G^{prf}_{\mathsf{F}}(A_f)$, so $\mathbf{Adv}^{prf}_{\mathsf{F}}(A_f) = \Pr[G_3] - \Pr[G_2]$.

Combining all the above equations gives the desired bound on $\mathbf{Adv}^{ss}_{\mathsf{StE},\mathcal{L},\mathcal{S}}(A)$.

2.4 Partially Precomputed Joins

We demonstrate our framework from Section 2.3 in action with two SQL data types: JnDT and SjDT. The former only supports non-recursive join queries and is presented for the purpose of introducing partially precomputed (PP) join indexing. The latter allows recursive queries, cluster joins and equality selections, and demonstrates how OPX's techniques can be modified to use PP joins.

2.4.1 Indexing of Non-Recursive Joins

Join data type JnDT**.**

We define JnDT.Dom to contain (DB, α) such that DB is a database and α is the set of join queries supported (i.e. if A is the set of attributes in DB that are not unique key attributes, then $\alpha \subseteq \{(at_1, at_2) \in A \times A : \mathsf{getID}(at_1, \mathsf{DB}) \neq \mathsf{getID}(at_2, \mathsf{DB})\}$. Our goal here is to capture SQL queries of the form "id_1 **join** id_2 **on** $at_1 = at_2$" where $id_1, id_2 \in \mathsf{DB.IDs}$ and $at_i \in \mathsf{DB}[id_i].\mathsf{Ats}$.

We allow queries to be any pair of attributes (i.e. $\mathsf{JnDT.QS} = \{(at_1, at_2) : at_i \in \{0,1\}^*\}$),

Alg $\mathcal{S}(lk^{\mathsf{i}}, N, L)$

$(\mathsf{IX}, (\mathsf{tk}_1, \ldots, \mathsf{tk}_n)) \leftarrow \mathcal{S}^{\mathsf{i}}(lk^{\mathsf{i}})$
$P \leftarrow \bigcup_{i \in [n]} \mathsf{StI.Eval}(\mathsf{tk}_i, \mathsf{IX})$
For $\mathsf{rt} \in \bigcup_{\mathbf{rt} \in P} \mathbf{rt}$ do $\mathbf{D}'[\mathsf{rt}] \leftarrow\$ \{0,1\}^{\mathsf{SE.cl}(L)}$
While $|\mathbf{D}'.\mathsf{Lbls}| < N$ do
  $\mathsf{rt} \leftarrow\$ \{0,1\}^{\mathsf{F.ol}}$; $\mathbf{D}'[\mathsf{rt}] \leftarrow\$ \{0,1\}^{\mathsf{SE.cl}(L)}$
Return $\big((\mathsf{IX}, \mathbf{D}'), (\mathsf{tk}_1, \ldots, \mathsf{tk}_n)\big)$

Subroutine $\mathsf{EncRows}^{\mathrm{Enc},\mathrm{Fn}}((\mathsf{DB}, \alpha))$

For $(id, \mathsf{R}) \in \mathsf{DB}$ do
  For $\mathbf{r} \in \mathsf{R.T}$ do $\mathbf{D}[(id, \mathbf{r}[\mathsf{uk}(id)])] \leftarrow \mathbf{r}$
Pad all values in $\mathbf{D}$ to the same length
For $\ell \in \mathbf{D}.\mathsf{Lbls}$ do
  $\mathbf{T}[\ell] \leftarrow\$ \mathrm{Fn}(\ell)$; $\mathbf{D}'[\mathbf{T}[\ell]] \leftarrow\$ \mathrm{Enc}(\mathbf{D}[\ell])$
Return $(\mathbf{D}', \alpha, \mathbf{T})$

Adversary $A_{\mathrm{i}}(\mathtt{s})$

$(\mathbf{DB}, \mathbf{q}, st) \leftarrow\$ A(\mathtt{s})$
$K_{\mathrm{e}} \leftarrow\$ \mathsf{SE.KS}$; $K_{\mathrm{f}} \leftarrow\$ \mathsf{F.KS}$
Define $\mathrm{Enc} : \mathrm{Enc}(x) = \mathsf{SE.Enc}(K_{\mathrm{e}}, \cdot)$
Define $\mathrm{Fn} : \mathrm{Fn}(x) = \mathsf{F.Ev}(K_{\mathrm{f}}, \cdot)$
$(\mathbf{D}', \alpha, \mathbf{T}) \leftarrow\$ \mathsf{EncRows}^{\mathrm{Enc},\mathrm{Fn}}(\mathbf{DB})$
Return $\big((\mathbf{D}', \alpha, \mathbf{T}), \mathbf{q}, (\mathbf{D}', st)\big)$

Adversary $A_{\mathrm{i}}\big(\mathsf{g}, \mathsf{IX}, \mathbf{tk}, (\mathbf{D}', st)\big)$

$b' \leftarrow\$ A\big(\mathsf{g}, (\mathsf{IX}, \mathbf{D}'), \mathbf{tk}, st\big)$
Return b'

Adversaries $\boxed{A_{\mathrm{s}}^{\mathrm{Enc}}}, \boxed{A_{\mathrm{f}}^{\mathrm{Fn}}}$

$(\mathbf{DB}, \mathbf{q}, st) \leftarrow\$ A(\mathtt{s})$; $K_{\mathrm{f}} \leftarrow\$ \mathsf{F.KS}$
$\boxed{\text{Define } \mathrm{Fn} : \mathrm{Fn}(x) = \mathsf{F.Ev}(K_{\mathrm{f}}, \cdot)}$
$\boxed{\text{Let } \mathrm{Enc} : \{0,1\}^L \to \{0,1\}^{\mathsf{SE.cl}(L)} \text{ be a random function}}$
$(\mathbf{D}', \alpha, \mathbf{T}) \leftarrow\$ \mathsf{EncRows}^{\mathrm{Enc},\mathrm{Fn}}(\mathbf{DB})$
$lk^{\mathsf{i}} \leftarrow\$ \mathcal{L}^{\mathsf{i}}(\mathsf{DS}, \mathbf{q})$
$(\mathsf{IX}, (\mathsf{tk}_1, \ldots, \mathsf{tk}_n)) \leftarrow \mathcal{S}^{\mathsf{i}}(lk^{\mathsf{i}})$
$b' \leftarrow\$ A\big(\mathsf{g}, (\mathsf{IX}, \mathbf{D}'), (\mathsf{tk}_1, \ldots, \mathsf{tk}_n), st\big)$
Return b'

Games $\boxed{\mathrm{G}_0(A)}, \boxed{\mathrm{G}_1(A)}$

$(\mathbf{DB}, \mathbf{q}, st) \leftarrow\$ A(\mathtt{s})$
$K_{\mathrm{e}} \leftarrow\$ \mathsf{SE.KS}$; $K_{\mathrm{f}} \leftarrow\$ \mathsf{F.KS}$
Define $\mathrm{Enc} : \mathrm{Enc}(x) = \mathsf{SE.Enc}(K_{\mathrm{e}}, \cdot)$
Define $\mathrm{Fn} : \mathrm{Fn}(x) = \mathsf{F.Ev}(K_{\mathrm{f}}, \cdot)$
$(\mathbf{D}', \alpha, \mathbf{T}) \leftarrow\$ \mathsf{EncRows}^{\mathrm{Enc},\mathrm{Fn}}(\mathbf{DB})$
$\boxed{\begin{array}{l} K_{\mathrm{i}}' \leftarrow\$ \mathsf{StI.KS} \\ (K_{\mathrm{i}}, \mathsf{IX}) \leftarrow\$ \mathsf{StI.Enc}(K_{\mathrm{i}}', \mathsf{DS}) \\ \text{For } i \in [n] \text{ do } \mathsf{tk}_i \leftarrow\$ \mathsf{StI.Tok}(K_{\mathrm{i}}, q_i) \end{array}}$
$\boxed{\begin{array}{l} lk^{\mathsf{i}} \leftarrow\$ \mathcal{L}^{\mathsf{i}}(\mathsf{DS}, (q_1, \ldots, q_n)) \\ (\mathsf{IX}, (\mathsf{tk}_1, \ldots, \mathsf{tk}_n)) \leftarrow \mathcal{S}^{\mathsf{i}}(lk^{\mathsf{i}}) \end{array}}$
$b' \leftarrow\$ A\big(\mathsf{g}, (\mathsf{IX}, \mathbf{D}'), (\mathsf{tk}_1, \ldots, \mathsf{tk}_n), st\big)$
Return $b' = 1$

Games $\boxed{\mathrm{G}_2(A)}, \boxed{\mathrm{G}_3(A)}$

$(\mathbf{DB}, \mathbf{q}, st) \leftarrow\$ A(\mathtt{s})$; $K_{\mathrm{f}} \leftarrow\$ \mathsf{F.KS}$
Let $\mathrm{Enc} : \{0,1\}^L \to \{0,1\}^{\mathsf{SE.cl}(L)}$ be a random function
$\boxed{\text{Define } \mathrm{Fn} : \mathrm{Fn}(x) = \mathsf{F.Ev}(K_{\mathrm{f}}, \cdot)}$
$\boxed{\text{Let } \mathrm{Fn} : \{0,1\}^* \to \{0,1\}^{\mathsf{F.ol}} \text{ be a random function}}$
$(\mathbf{D}', \alpha, \mathbf{T}) \leftarrow\$ \mathsf{EncRows}^{\mathrm{Enc},\mathrm{Fn}}(\mathbf{DB})$
$lk^{\mathsf{i}} \leftarrow\$ \mathcal{L}^{\mathsf{i}}(\mathsf{DS}, (q_1, \ldots, q_n))$
$(\mathsf{IX}, (\mathsf{tk}_1, \ldots, \mathsf{tk}_n)) \leftarrow \mathcal{S}^{\mathsf{i}}(lk^{\mathsf{i}})$
$b' \leftarrow\$ A\big(\mathsf{g}, (\mathsf{IX}, \mathbf{D}'), (\mathsf{tk}_1, \ldots, \mathsf{tk}_n), st\big)$
Return $b' = 1$

Figure 2.6. Simulator, adversaries and games used in the proof of Theorem 13.

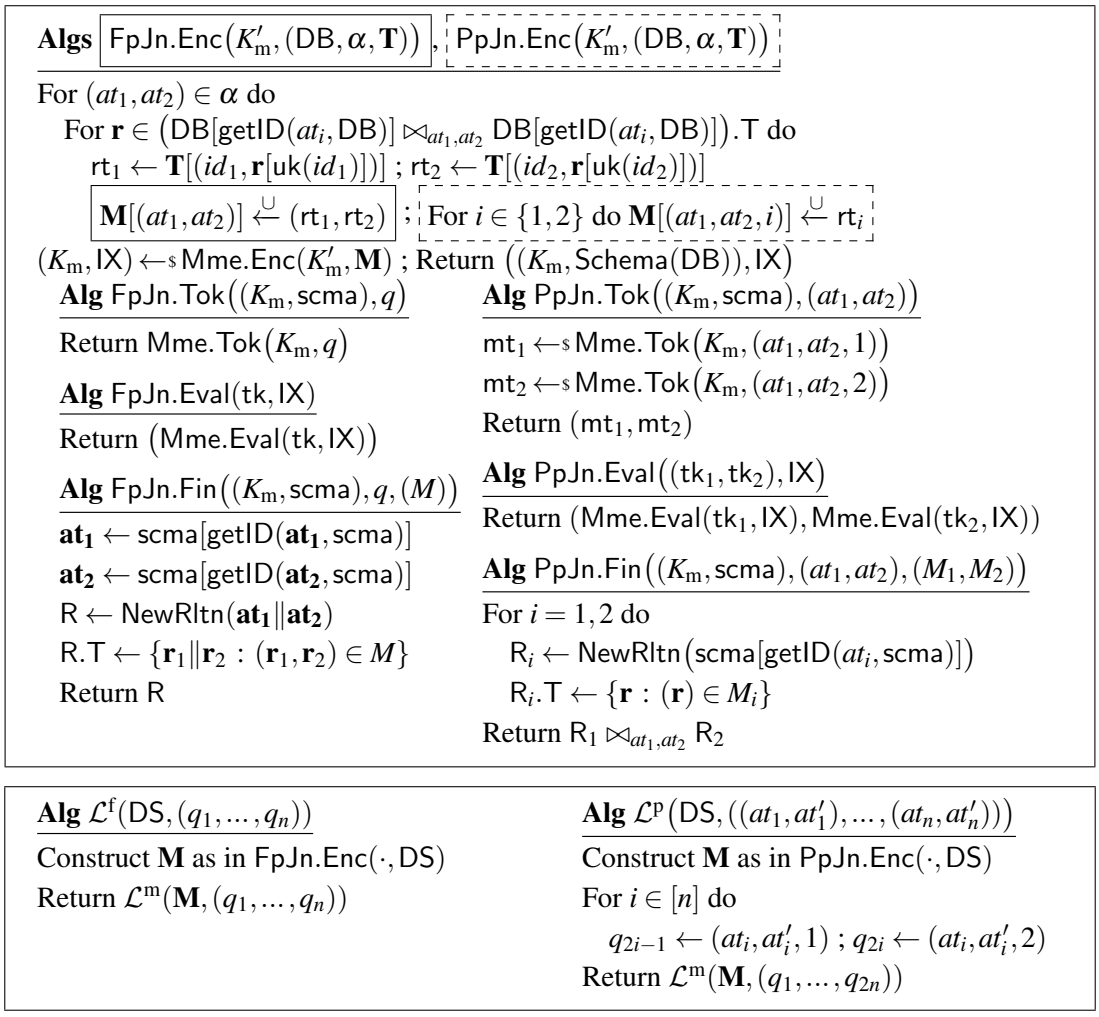

Figure 2.7. Algorithms of StI schemes FpJn, PpJn (top) and their leakage algorithms (bottom) where Mme is a RR multimap encryption scheme. Note that in the encryption algorithm, boxed code belongs only to the respective algorithm.

but JnDT.Spec only computes the join if $(at_1,at_2)\in\alpha$:

$$\mathsf{JnDT.Spec}\big((at_1,at_2),(\mathsf{DB},\alpha)\big)=\mathsf{DB}[\mathsf{getID}(at_1,\mathsf{DB})]\bowtie_{at_1,at_2}\mathsf{DB}[\mathsf{getID}(at_2,\mathsf{DB})]$$

and returns $\perp$ otherwise. From here on we assume that all queries made are "non-trivial" meaning they return relations with at least one row.

FP indexing.

FpJn is an StI scheme that "fully precomputes" joins and is modeled after SPX's handling

of "type-2 selections" and OPX's handling of "leaf joins" [89, 92]. The intuition here is that the output relation for each possible join query is precomputed and pointers to the rows therein are stored as an entry in a RR encrypted multimap. FpJn's detailed algorithms and leakage profile are given in Fig. 2.7. Note that $\mathsf{FpJn.KS} = \mathsf{Mme.KS}$ and that each row in the output of a particular join is indexed as a pair of pointers to rows in DB.

Since join queries are handled directly by Mme the leakage and efficiency of FpJn depends entirely on Mme. For the rest of this discussion, we will assume Mme is one of the mainstream multimap encryption schemes (e.g. [48, 56, 52]) with the "standard" leakage profile consisting the label space size $|\mathbf{M}.\mathsf{Lbls}|$, multimap size $\sum_{\ell \in \mathbf{M}.\mathsf{Lbls}} |\mathbf{M}[\ell]|$, query pattern (equality pattern of queries $\ell_1, \ldots, \ell_n$) and query responses $\mathbf{M}[\ell_1], \ldots, \mathbf{M}[\ell_n]$.

Notice that when a query (at_1, at_2) is made in FpJn, the query responses reveal the equality pattern of columns at_1, at_2 for rows that appear in the join output. To illustrate, if the query is made on $\mathsf{DB} = \{(id_1, \mathsf{R}_1), (id_2, \mathsf{R}_2)\}$ where $\mathsf{R}_1, \mathsf{R}_2$ are as depicted in Fig. 2.4, the server learns that the first two rows of each R_i all have the same value in their at_1, at_2 columns, but won't reveal anything about the last two rows of each R_i apart from the fact that they are not returned in the join. Note that in the worst case, the join returns all rows from both relations and the search pattern leakage reveals the entire equality pattern of both columns. This leakage is comparable to PRE-based techniques like deterministic encryption or adjustable joins (an observation also made by DPPS [59]). This is significant because, as discussed in Section 2.1, LAAs are highly effective against PRE and can be applied in this case. Beyond the worst case, FP indexing leaks strictly less than PRE-based solutions but this does not make them immune to LAAs. In particular, we believe that attacks (such as those using ℓ_p-optimization or graph matching [110, 35]) can be extended to make use of partial equality patterns and cross column correlations, and be effective against FpJn's leakage.

We also note that FpJn achieves lower leakage than the analogous indexing in SPX or OPX because it uses a single multimap. The latter schemes had one encrypted multimap for each attribute (i.e. $\mathbf{M}_{at_1}$ indexes all joins $(at_1, at_2) \in \alpha$) this leaks additional metadata and tells the

adversary when two queries join on the same at_1.

PP indexing.

We introduce a new StI scheme PpJn which performs "partially precomputed" indexing, whose algorithms are also depicted in Fig. 2.7. PpJn.Enc proceeds in the same way as FpJn.Enc but we store the rows from each input relation separately. In other words, if $\mathbf{M}_\mathrm{f}, \mathbf{M}_\mathrm{p}$ are the multimaps constructed in the respective setup algorithms, then $\mathbf{M}_\mathrm{p}[(at_1, at_2, i)] = \{\mathsf{rt}_i : (\mathsf{rt}_1, \mathsf{rt}_2) \in \mathbf{M}_\mathrm{f}[(at_1, at_2)]\}$ for $i = 1, 2$ and $(at_1, at_2) \in \alpha$. Notice that this means the client needs to reassemble the output relation from the two sets of rows in StI.Fin. We recommend that the client do so by sorting then joining the columns, avoiding the quadratic time nested loop join where rows are compared pairwise.

This small change in indexing technique has substantial impact on bandwidth and security. In the worst case, the number of rows sent with FP is quadratic while PP's is linear. This bandwidth reduction occurs because two sets of rows are sent instead of their cross product. Notice that modulo some metadata information (i.e. the multimap sizes), the PP leakage can be derived from the FP leakage meaning that PP indexing is no worse than FP indexing. In fact, if more than one row is returned to any query PP leakage is strictly lower. To illustrate, when join query (at_1, at_2) is made to the aforementioned database in Fig. 2.4, the adversary sees that the first three rows of both tables were returned and can infer that each row has at least one matching value in the other column – nothing specific about their equality patterns.

In summary, PpJn is the superior indexing choice for JnDT because its leakage is strictly lower, bandwidth is no worse and efficiency is comparable.

Semantic security.

The security of FpJn, PpJn reduce to that of Mme. The proof follows directly from the definition of Mme's semantic security.

Theorem 14 *Let $\mathcal{L}, \mathcal{S}$ be the leakage algorithm and simulator for* Mme. *Let $\mathcal{L}^\mathrm{f}, \mathcal{L}^\mathrm{p}$ be the*

Adversary $A_{\mathrm{m}}(\mathtt{s})$ Return $A(\mathtt{s})$ **Adversary** $A_{\mathrm{m}}(\mathsf{g},\mathsf{IX},(\mathsf{tk}_1,\ldots,\mathsf{tk}_{2n}),st)$ $\mathbf{tk} \leftarrow \big((\mathsf{tk}_1,\mathsf{tk}_2),\ldots,(\mathsf{tk}_{2n-1},\mathsf{tk}_{2n})\big)$ $b' \leftarrow\!\!\$\, A(\mathsf{g},\mathsf{IX},\mathbf{tk},st)$; Return b'	**Alg** $\mathcal{S}^{\mathrm{p}}(lk^{\mathrm{m}})$ $(\mathsf{EM},(\mathsf{tk}_1,\ldots,\mathsf{tk}_{2n})) \leftarrow \mathcal{S}^{\mathrm{m}}(lk^{\mathrm{m}})$ $\mathbf{tk} \leftarrow \big((\mathsf{tk}_1,\mathsf{tk}_2),\ldots,(\mathsf{tk}_{2n-1},\mathsf{tk}_{2n})\big)$ Return $(\mathsf{EM},\mathbf{tk})$

Figure 2.8. Simulators (right) and adversaries (left) used in the proof of Theorem 14.

leakage algorithms given in Fig. 2.7. Then, given adversary A these exists adversary A_{m} *and simualtor* $\mathcal{S}^{\mathrm{p}}$ *such that:*

$$\mathbf{Adv}^{\mathrm{ss}}_{\mathsf{FpJn},\mathcal{L}^{\mathrm{f}},\mathcal{S}}(A) \leq \mathbf{Adv}^{\mathrm{ss}}_{\mathsf{Mme},\mathcal{L},\mathcal{S}}(A) \text{ and } \mathbf{Adv}^{\mathrm{ss}}_{\mathsf{PpJn},\mathcal{L}^{\mathrm{p}},\mathcal{S}^{\mathrm{p}}}(A) \leq \mathbf{Adv}^{\mathrm{ss}}_{\mathsf{Mme},\mathcal{L},\mathcal{S}}(A_{\mathrm{m}}).$$

Proof. The first result follows directly from the definition of $\mathsf{FpJn},\mathcal{L}^{\mathrm{f}}$. The second result requires us to define $A_{\mathrm{m}},\mathcal{S}^{\mathrm{p}}$, which we do in Fig. 2.8. In both of these, tokens are just concatenated and deconcatenated as needed by the definition of PpJn. The result follows immediately.

2.4.2 PP indexing for recursive queries

SjDT data type.

We expand the query support of JnDT to include equality selections, cluster joins (joins on more than one attribute) and recursively defined queries. The resultant query class is similar to the SPJ algebra defined by CM [51] except for the omission of the projection operation which we note can be handled as a post-processing step requiring no cryptographic techniques.

We capture this via the SQL data type SjDT. Its domain is unchanged from JnDT.Dom except that α allows tuple pairs in addition to attribute pairs. SjDT.QS's queries are recursively defined and can be divided into three types. Below, we describe these as well as their evaluation and SQL equivalent. These are defined recursively so $q_i, \mathbf{q}_i$ are themselves SjDT and SQL queries respectively:

- **Relation retrieval queries:** These are queries $(\mathtt{r}, id) \in \mathsf{SjDT.QS}$ which model SQL queries of the form "**select** * **from** *id*". These are evaluated as follows:

$$\mathsf{SjDT.Spec}\big((\mathtt{r}, id), \mathbf{DB}\big) = \mathsf{DB}[id] \text{ where } \mathbf{DB} = (\mathsf{DB}, \alpha).$$

- **(Equality) selections queries:** These are queries $(\mathtt{s}, at, x, q_1) \in \mathsf{SjDT.QS}$ which model SQL queries of the form "**select** * **from** $\mathbf{q}_1$ **where** $at = c$". These are evaluated as follows:

$$\mathsf{SjDT.Spec}\big((\mathtt{s}, at, x, q_1), \mathbf{DB}\big) = \sigma_{(at,x)}(\mathsf{SjDT.Spec}(q_1, \mathbf{DB})).$$

- **(Equi)joins queries:** These are queries $(\mathtt{j}, \mathbf{t}_1, \mathbf{t}_2, q_1, q_2) \in \mathsf{SjDT.QS}$ which model SQL queries of the form "**select** * **from** $\mathbf{q}_1$ **join** $\mathbf{q}_2$ **on** $\mathbf{t}_1 = \mathbf{t}_2$". These are evaluated as follows:

$$\mathsf{SjDT.Spec}\big((\mathtt{j}, \mathbf{t}_1, \mathbf{t}_2, q_1, q_2), \mathbf{DB}\big) = (\mathsf{SjDT.Spec}(q_1, \mathbf{DB})) \bowtie_{\mathbf{t}_1, \mathbf{t}_2} (\mathsf{SjDT.Spec}(q_2, \mathbf{DB})).$$

We say that queries of the form $(\mathtt{r}, id)$, $(\mathtt{s}, at, x, (\mathtt{r}, id))$ or $(\mathtt{j}, \mathbf{t}_1, \mathbf{t}_2, (\mathtt{r}, id_1), (\mathtt{r}, id_2))$ are *non-recursive* and all others are *recursive*. We require that all attributes in each $\mathbf{t}_i$ come from the same relation in DB (i.e. $\mathbf{t}_i \subseteq \mathsf{DB}[id].\mathsf{Ats}$ for some $id \in \mathsf{DB.IDs}$). While allowing cluster joins may lead to an exponential-size index, a judicious database administrator would not allow this – cluster joins are rarely used and usually known in advance.

Hashset filtering.

To minimize the leakage of recursive queries in our StI schemes we employ the filtering hashset technique introduced in OXT [49]. We now review this technique and establish some notation for it.

This filtering hashset is a set denoted HS containing outputs of a function family F where $\mathsf{F.KS} = \{0,1\}^{\mathsf{F.ol}}$. In our algorithms, the hashset will be used to associate predicate bitstrings

with a row tokens (from $\mathbf{T}$). Later, given a predicate p's key $K = \mathsf{F.Ev}(K_\mathrm{f}, p)$ we can filter a set of row tokens, retaining only those which satisfy the predicate. We formalize this via the following algorithms:

$\underline{\textbf{Alg } \mathsf{HsEnc}(K_\mathrm{f}, \mathsf{SET})}$	$\underline{\textbf{Alg } \mathsf{HsFilter}(K, (P_1, \ldots, P_n), \mathsf{HS})}$
For $(p, \mathsf{rt}) \in \mathsf{SET}$ do	For $i \in [n]$ do
$\quad \mathsf{HS} \xleftarrow{\cup} \mathsf{F.Ev}\big(\mathsf{F.Ev}(K_\mathrm{f}, p), \mathsf{rt}\big)$	$\quad$ For $\mathsf{rt} \in \mathbf{rt} \in P_i$ if $\mathsf{F.Ev}(K, \mathsf{rt}) \in \mathsf{HS}$ then $S \xleftarrow{\cup} \mathbf{rt}$
Return HS	$\quad$ If $S \neq \emptyset$ then $P_i \leftarrow S$
	Return $(P_1, \ldots, P_n)$

For notational convenience in our pseudocodes, $\mathsf{HsFilter}$ takes as input a tuple set $P = (P_1, \ldots, P_n)$. It then attempts to filter each P_i and retain only the tuples where at least one rt satisfies the predicate. However, if no such tuple exists, it does not perform the filtering at all.

PP indexing for SjDT.

We are now ready to extend the PP indexing technique introduced in Section 2.4 to construct StI for SjDT. On a high level, we do so by using an inverted index (similar to those used for SSE) to handle selections and a filtering hashset to handle recursively defined queries. The result is StI scheme PpSj whose algorithms are depicted in Fig. 2.9.

Now we provide some intuition for PpSj's algorithms. The scheme has two server-side data structures: an encrypted multimap and a hashset. The multimap is used to index non-recursive queries my mapping a query-derived label to the relevant rows in the database. For example, the label for relation retrieval query $(\mathtt{r}, id)$ is the query itself and its values are row tokens associated to rows in $\mathsf{DB}[id]$ (i.e. $\{(\mathbf{T}[(id, \mathbf{r}[\mathsf{uk}(id)])]) : \mathbf{r} \in \mathsf{DB}[id].\mathsf{T}\}$). Note that the latter are singleton tuples because we required that pointer tuples be made out of tuples of tokens. The hashset is used to filter the sets in a pointer tuple during a recursive query. For example, when processing the query $(\mathtt{j}, \mathbf{t}_1, \mathbf{t}_2, (\mathtt{s}, at, x, (\mathtt{r}, id_1)), (\mathtt{r}, id_2))$ (a select followed by a join), the server would use the multimap to retrieve row tokens for each of the non-recursive subqueries (i.e. $(\mathtt{s}, at, x, (\mathtt{r}, id_1))$ and $(\mathtt{r}, id_2)$). The token would also include two keys which can be used with $\mathsf{HsFilter}$ which tests if the rows being pointed to (in $\mathsf{DB}[id_1]$ or $\mathsf{DB}[id_2]$) are in

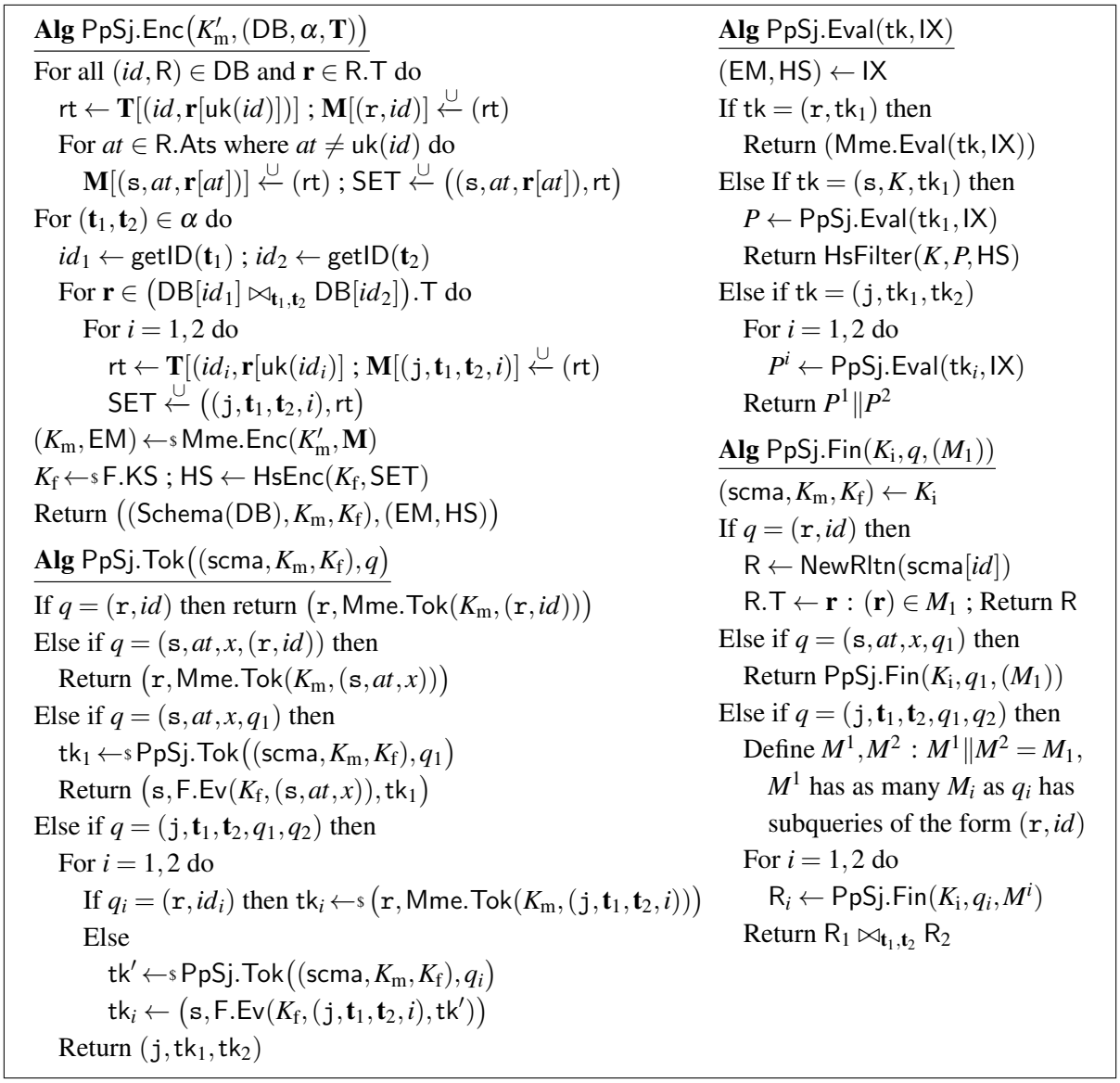

Figure 2.9. Algorithms for PpSj the StI scheme for SjDT using PP indexing.

$\mathsf{DB}[id_1] \bowtie_{\mathbf{t}_1,\mathbf{t}_2} \mathsf{DB}[id_2]$.

FP indexing for SjDT.

We analogously extend FpJn introduced in Section 2.4 to construct FpSj, an StI for SjDT. Just like with PpSj, non-recursive queries will be added to the encrypted multimap that is used to index the non-recursive joins while all recursive queries are filtered using the hashset. The only subtlety in this extension is the handling of "internal joins" which are queries of the form $q = (\mathtt{j}, \mathbf{t}_1, \mathbf{t}_2, (\mathtt{r}, id), q_1)$ (or $q = (\mathtt{j}, \mathbf{t}_1, \mathbf{t}_2, q_1, (\mathtt{r}, id))$) because we want to limit the row tokens leaked from *id* to those who join with some row returned by q_1. Similar to OPX, we construct an index where each row token returned in the subquery will "point to" the tokens of the rows joined to in $\mathsf{DB}[id]$. As alluded to in Section 2.3.2, this self-referential indexing (where Mme tokens are stored in $\mathbf{M}$) may introduce circular security issues if pathological Mme primitives are used. We avoid this by indexing internal joins with a specific, non-pathological primitive (as was done in OPX). To avoid the increased leakage and complexity of an additional data structure, we will assume that Mme is the $\mathsf{Mme}^{\mathrm{rr}}_{\pi}$ scheme recounted in Section 2.3 and co-locate this index with the one used for non-recursive queries. Notice that this subtlety does not come up in PpSj because we do not reveal join equality patterns so all recursive joins can be handled similarly.

The resultant StI scheme is FpSj whose algorithms are depicted in Fig. 2.10. Notice that $\mathsf{FpSj.KS} = \mathsf{Mme}^{\mathrm{rr}}_{\pi}.\mathsf{KS} = \mathsf{F.KS}$. The flags $\mathtt{iij}, \mathtt{ij}$ used come from the terms used for query classification by KMZZ in [92] where recursive joins are split into "internal joins" (i.e. queries of the form $(\mathtt{j}, \mathbf{t}_1, \mathbf{t}_2, (\mathtt{r}, id_1), q_2)$ or $(\mathtt{j}, \mathbf{t}_1, \mathbf{t}_2, (\mathtt{r}, id_1), q_2)$ and "intermediate internal joins" (i.e.those of form $(\mathtt{j}, \mathbf{t}_1, \mathbf{t}_2, q_1, q_2)$). We handle the internal joins described above by manually adding entries to the server-side data structure of $\mathsf{Mme}^{\mathrm{rr}}_{\pi}$ (i.e. dictionary $\mathbf{D}$) to index them.

We note that the StE scheme $\mathsf{StE} = \mathbf{SqlStE}[\mathsf{FpSj}, \mathsf{SE}, \mathsf{F}]$ is essentially the same as OPX with minor improvements in leakage (analogous to those described in our discussion of FpJn in Section 2.4) and a slightly revised approach to "internal joins".

Alg $\mathsf{FpSj.Enc}\big(K_{\mathrm{m}},(\mathsf{DB},\alpha,\mathbf{T})\big)$

For all $(id,\mathsf{R})\in\mathsf{DB}$ and $\mathbf{r}\in\mathsf{R.T}$ do
  $\mathsf{rt}\leftarrow\mathbf{T}[(id,\mathbf{r}[\mathsf{uk}(id)])]$; $\mathbf{M}[(\mathtt{r},id)]\xleftarrow{\cup}(\mathsf{rt})$
  For $at\in\mathsf{R.Ats}$ where $at\neq\mathsf{uk}(id)$ do
    $\mathbf{M}[(\mathtt{s},at,\mathbf{r}[at])]\xleftarrow{\cup}(\mathsf{rt})$; $\mathsf{SET}\xleftarrow{\cup}\big((\mathtt{s},at,\mathbf{r}[at]),\mathsf{rt}\big)$
For $(\mathbf{t}_1,\mathbf{t}_2)\in\alpha$ do
  $id_1\leftarrow\mathsf{getID}(\mathbf{t}_1)$; $id_2\leftarrow\mathsf{getID}(\mathbf{t}_2)$
  For $\mathbf{r}\in\big(\mathsf{DB}[id_1]\bowtie_{\mathbf{t}_1,\mathbf{t}_2}\mathsf{DB}[id_2]\big).\mathsf{T}$ do
    For $i=1,2$ do $\mathsf{rt}_i\leftarrow\mathbf{T}[(id_i,\mathbf{r}[\mathsf{uk}(id_i)])]$
    $\mathbf{M}[(\mathtt{j},\mathbf{t}_1,\mathbf{t}_2)]\xleftarrow{\cup}(\mathsf{rt}_1,\mathsf{rt}_2)$; $\mathbf{M}_1[(\mathbf{t}_1,\mathbf{t}_2,\mathsf{rt}_1,1)]\xleftarrow{\cup}\mathsf{rt}_2$
    $\mathbf{M}_1[(\mathbf{t}_1,\mathbf{t}_2,\mathsf{rt}_2,2)]\xleftarrow{\cup}\mathsf{rt}_1$
    $\mathsf{SET}\xleftarrow{\cup}\big((\mathtt{iij},\mathbf{t}_1,\mathbf{t}_2),(\mathsf{rt}_1,\mathsf{rt}_2)\big)$
$(K_{\mathrm{m}},\mathbf{D})\leftarrow\!\!\$\,\mathsf{Mme}^{\mathrm{rr}}_{\pi}.\mathsf{Enc}(K_{\mathrm{m}},\mathbf{M})$
For $(\mathbf{t}_1,\mathbf{t}_2,\mathsf{rt},i)\in\mathbf{M}_1.\mathsf{Lbls}$ do
  For $j=0,1$ do $K_j\leftarrow\mathsf{F.Ev}(\mathsf{F.Ev}(K_{\mathrm{m}},(\mathtt{ij},\mathbf{t}_1,\mathbf{t}_2,i)),\mathsf{rt}\|j)$
  $\{\mathsf{rt}_1,\ldots,\mathsf{rt}_n\}\leftarrow\mathbf{M}_1[(\mathbf{t}_1,\mathbf{t}_2,\mathsf{rt},i)]$
  For $k\in[n]$ do
    Pad rt_k to $\mathbf{M}$'s max. value length
    $\mathbf{D}[\mathsf{F.Ev}(K_0,k)]\leftarrow\!\!\$\,\mathsf{SE.Enc}(K_1,\mathsf{rt}_k)$
$K_{\mathrm{f}}\leftarrow\!\!\$\,\mathsf{F.KS}$; $\mathsf{HS}\leftarrow\mathsf{HsEnc}(K_{\mathrm{f}},\mathsf{SET})$
Return $\big((\mathsf{Schema}(\mathsf{DB}),K_{\mathrm{m}},K_{\mathrm{f}}),(\mathbf{D},\mathsf{HS})\big)$

Alg $\mathsf{FpSj.Tok}(K_{\mathrm{i}},q)$

$(\mathsf{scma},K_{\mathrm{m}},K_{\mathrm{f}})\leftarrow K_{\mathrm{i}}$
If $q=(\mathtt{r},id)$ then return $(\mathtt{r},\mathsf{Mme}^{\mathrm{rr}}_{\pi}.\mathsf{Tok}(K_{\mathrm{m}},(\mathtt{r},id)))$
Else if $q=(\mathtt{s},at,x,(\mathtt{r},id))$ then
  Return $\big(\mathtt{r},\mathsf{Mme}^{\mathrm{rr}}_{\pi}.\mathsf{Tok}(K_{\mathrm{m}},(\mathtt{s},at,x))\big)$
Else if $q=(\mathtt{s},at,x,q_1)$ then
  Return $\big(\mathtt{s},\mathsf{F.Ev}(K_{\mathrm{f}},(\mathtt{s},at,x)),\mathsf{FpSj.Tok}(K_{\mathrm{i}},q_1)\big)$
Else if $q=(\mathtt{j},\mathbf{t}_1,\mathbf{t}_2,(\mathtt{r},id_1),(\mathtt{r},id_2))$ then
  Return $\big(\mathtt{r},\mathsf{Mme}^{\mathrm{rr}}_{\pi}.\mathsf{Tok}(K_{\mathrm{m}},(\mathtt{j},\mathbf{t}_1,\mathbf{t}_2))\big)$
Else if $q=(\mathtt{j},\mathbf{t}_1,\mathbf{t}_2,q_1,(\mathtt{r},id))$ then
  Return $(\mathtt{ij},\mathsf{F.Ev}(K_{\mathrm{m}},(\mathtt{ij},\mathbf{t}_1,\mathbf{t}_2,1)),1,\mathsf{FpSj.Tok}(K_{\mathrm{i}},q_1))$
Else if $q=(\mathtt{j},\mathbf{t}_1,\mathbf{t}_2,(\mathtt{r},id),q_1)$ then
  Return $(\mathtt{ij},\mathsf{F.Ev}(K_{\mathrm{m}},(\mathtt{ij},\mathbf{t}_1,\mathbf{t}_2,2)),2,\mathsf{FpSj.Tok}(K_{\mathrm{i}},q_1))$
Else if $q=(\mathtt{j},\mathbf{t}_1,\mathbf{t}_2,q_1,q_2)$ then
  For $i=1,2$ do $\mathsf{tk}_i\leftarrow\!\!\$\,\mathsf{FpSj.Tok}(K_{\mathrm{i}},q_i)$
  Return $\big(\mathtt{iij},\mathsf{F.Ev}(K_{\mathrm{f}},(\mathtt{iij},\mathbf{t}_1,\mathbf{t}_2)),\mathsf{tk}_1,\mathsf{tk}_2\big)$

Alg $\mathsf{FpSj.Eval}(\mathsf{tk},(\mathbf{D},\mathsf{HS}))$

If $\mathsf{tk}=(\mathtt{r},\mathsf{mt})$ then
  Return $(\mathsf{Mme}^{\mathrm{rr}}_{\pi}.\mathsf{Eval}(\mathsf{mt},\mathbf{D}))$
Else if $\mathsf{tk}=(\mathtt{s},K,\mathsf{tk}_1)$ then
  $P\leftarrow\mathsf{FpSj.Eval}(\mathsf{tk}_1,(\mathbf{D},\mathsf{HS}))$
  Return $\mathsf{HsFilter}(K,P,\mathsf{HS})$
Else if $\mathsf{tk}=(\mathtt{ij},K,i,\mathsf{tk}_1)$ then
  $(P_1)\leftarrow\mathsf{FpSj.Eval}(\mathsf{tk}_1,(\mathbf{D},\mathsf{HS}))$
  For $\mathsf{rt}\in\mathbf{rt}\in P_1$ do
    For $j=0,1$ do
      $K_j\leftarrow\mathsf{F.Ev}(K,\mathsf{rt}\|j)$
    $x\leftarrow\mathbf{D}[\mathsf{F.Ev}(K_0,c_{\mathsf{rt}})]$
    While $x\neq\perp$ do
      $S\xleftarrow{\cup}(\mathsf{SE.Dec}(K_1,x),\mathbf{rt})$
      $c_{\mathsf{rt}}\leftarrow c_{\mathsf{rt}}+1$
      $x\leftarrow\mathbf{D}[\mathsf{F.Ev}(K_0,c_{\mathsf{rt}})]$
  If $i=1$ then
    $P_1\leftarrow\{(\mathsf{rt})\|\mathbf{rt}:(\mathsf{rt},\mathbf{rt})\in S\}$
  Else
    $P_1\leftarrow\{\mathbf{rt}\|(\mathsf{rt}):(\mathsf{rt},\mathbf{rt})\in S\}$
  Return (P_1)
Else if $\mathsf{tk}=(\mathtt{iij},K,\mathsf{tk}_1,\mathsf{tk}_2)$ then
  For $i=1,2$ do
    $(P_i)\leftarrow\mathsf{FpSj.Eval}(\mathsf{tk}_i,(\mathbf{D},\mathsf{HS}))$
  For $\mathbf{rt}_1\in P_1$ and $\mathbf{rt}_2\in P_2$ do
    For $\mathsf{rt}_1\in\mathbf{rt}_1$ and $\mathsf{rt}_2\in\mathbf{rt}_2$ do
      If $\mathsf{F.Ev}(K,(\mathsf{rt}_1,\mathsf{rt}_2))\in\mathsf{HS}$ then
        $P_0\xleftarrow{\cup}\mathbf{rt}_1\|\mathbf{rt}_2$
  Return (P_0)

Alg $\mathsf{FpSj.Fin}(K_{\mathrm{i}},q,(M_1))$

$(\mathsf{scma},K_{\mathrm{m}},K_{\mathrm{f}})\leftarrow K_{\mathrm{i}}$
Using scma and q, compute the
  attributes $\mathbf{at}$ in $\mathsf{SjDT.Spec}(q,\mathsf{DS})$
$\mathsf{R}\leftarrow\mathsf{NewRltn}(\mathbf{at})$
For $(m_1,\ldots,m_n)\in M_1$ do
  $\mathsf{R.T}\xleftarrow{\cup}m_1\|\ldots\|m_n$
Return R

Figure 2.10. Algorithms for FpSj, the StI scheme for SjDT using FP indexing.

PpSj **leakage discussion.**

While a pseudocode description of PpSj's leakage profile may seem convoluted, we believe the intuition behind it enables helpful comparisons with FpSj and OPX [92]. As such, we aim to give some intuition by describing the components of PpSj's leakage profile via a running example, before giving a full description of PpSj's leakage algorithm and the associated security proof. Below, we assume that MME primitives have the "standard" leakage profile (as described in Section 2.3).

Our example database contains $\mathsf{R}_1, \mathsf{R}_2$ from Fig. 2.4. If no queries are made, the server-side data structures reveal only *metadata leakage*. This includes the number of values in the multimap, the maximum-length of a value in the multimap and the number of F outputs in the hashset. The leakage of FpSj is comparable but on OPX it is higher because different data structures are used to index different SQL operations.

We will refer to all other forms of leakage as "query dependent leakage". This is where PP indexing has substantial savings over FP and OPX.

Now lets assume the client makes the following queries: $q_1 = (\mathtt{s}, at_3, \mathtt{CS}, (\mathtt{r}, id_2))$, $q_2 = (\mathtt{s}, at_2, \mathtt{Eve}, (\mathtt{r}, id_2))$, $q_3 = (\mathtt{r}, id_1)$ and $q_4 = \big(\mathtt{j}, at_1, at_2, (\mathtt{r}, id_1), (\mathtt{s}, at_3, \mathtt{CS}, (\mathtt{r}, id_2))\big)$. The server will receive four tokens, where $\mathsf{tk}_1, \mathsf{tk}_2, \mathsf{tk}_3$ are such that $\mathsf{tk}_i = (\mathtt{r}, \mathsf{mt}_i)$ and $\mathsf{tk}_4 = \big(\mathtt{j}, (\mathtt{r}, \mathsf{mt}_4), (\mathtt{j}, K, \mathsf{mt}_5)\big)$. Here, each mt_i is a token for Mme while K is a hashset key. Just from inspecting these, the adversary learns the *recursion structure* of the queries. Specifically, he learns that the first three queries were non-recursive while q_4 was a join followed by a select. This leakage is slightly lower in FpSj, PpSj compared to OPX because the adversary cannot differentiate between non-recursive selections and relation retrievals.

The Mme tokens leak *the multimap query pattern* and *multimap responses*. The former reveals whenever the associated query or subquery is repeated. In our case, the adversary learns that $\mathsf{mt}_1, \mathsf{mt}_5$ are associated to the same query. Note that this does not extend to $\mathsf{mt}_3, \mathsf{mt}_4$ because the latter is in a join. From the multimap query responses he "sees" the row tokens that are returned by each $\mathsf{Mme.Eval}(\mathsf{mt}_i, \mathsf{EM})$. This reveals the equality pattern of the rows returned

by each associated query/subquery. For example, this reveals that q_1, q_2 both return two rows, one of which is shared. On join queries, we enjoy similar leakage savings as described in the non-recursive case. For example, tk_4 will reveal that three rows are returned from the left relation (i.e. id_1) but doesn't say anything about whether they are in the final output relation or how they "match up" with rows from the right relation. In FpSj and OPX, both of the above are revealed.

Finally, the hashset keys reveal the *hashset key query pattern* and *hashset filtering results*. The former reveals when the exact same predicate is repeated and is detectable because the keys would be the same. The latter is because the adversary is free to apply hashset keys (in the tokens) to filter all the row tokens he can retrieve from EM thereby learning the *hashset filtering results*. This means that using K he can learn that one row returned by q_2 satisfies the predicate associated to K even though it is not in the output of q_4. Similarly, he learns that two rows returned by q_1 satisfies the predicate but only one is returned by q_4. Using FpSj the adversary would additionally learn which row returned in q_3 is "paired up" with this row in the q_4 output.

Leakage comparison.

From the above discussion, one might expect decreasing query-dependent leakage from PpSj to FpSj to OPX. While the leakage for FpSj can always be derived from OPX, the comparison of PpSj to FpSj is not as straightforward because they sometimes do not return the same rows when recursive queries are made (which we discuss in more detail below).

However, when restricted to non-recursive queries, PpSj's query-dependent leakage is strictly superior for the same reasons that PpJn was superior in Section 2.4. Extending this, we can upper bound the leakage lk_{p} of PpSj on queries $q_1, \ldots, q_n$ with its leakage lk'_{p} on the minimal set of non-recursive queries $q'_1, \ldots, q'_m$ with which the server can still deduce the pointer tuples it should return on $q_1, \ldots, q_n$. Doing the same for FpSj, we have $lk_{\mathrm{f}} \leq lk'_{\mathrm{f}}$ as well. Then, via the above observation about non-recursive queries we have $lk'_{\mathrm{p}} \leq lk'_{\mathrm{f}}$, with the inequality being strict if at least one join query with at least two rows is made. Our being able to *bound* PpSj*'s query-dependent leakage lower than* FpSj*'s* gives credence to the intuition that PpSj is the more

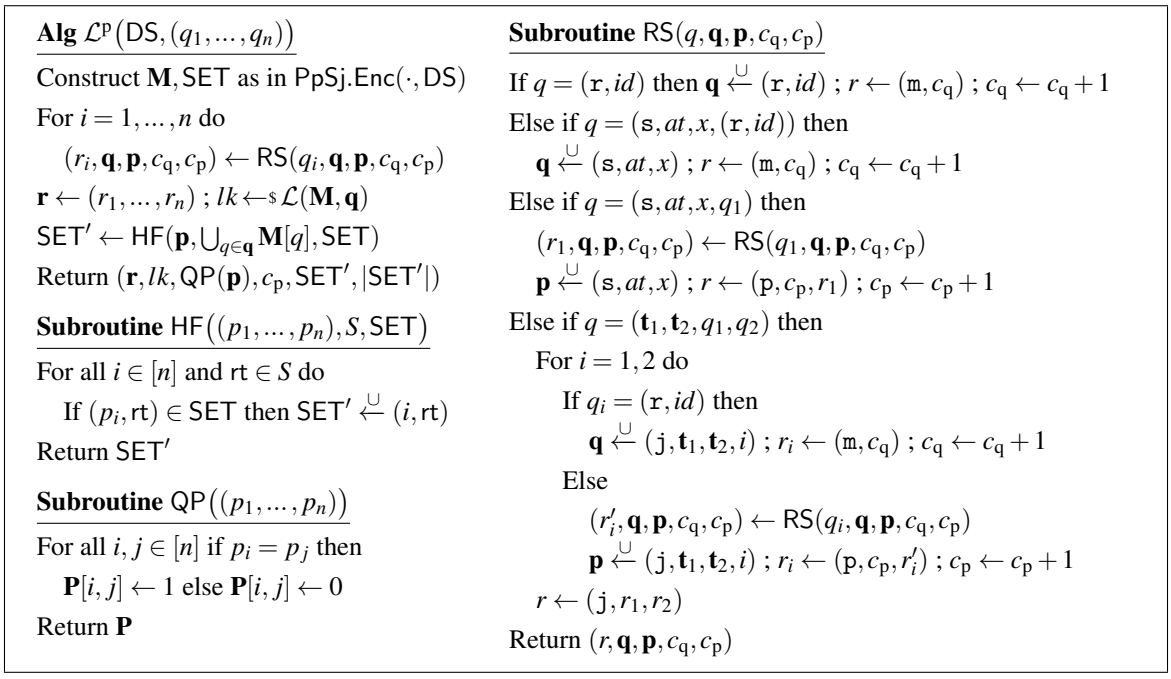

Figure 2.11. Leakage profile for PpSj where $\mathsf{RS},\mathcal{L},\mathsf{HF},\mathsf{QP}$ compute the recursion structure leakage, Mme's leakage profile, hashset filtering results and hashset query pattern respectively.

secure variant in practice.

Security Proofs.

For completeness, we now give the leakage algorithms associated to FpSj, PpSj, and their associated security proofs.

The leakage algorithm for PpSj is $\mathcal{L}^{\mathrm{p}}$ in Fig. 2.11. It calls the three subroutines which compute the query-dependent leakage. RS is first called on each of the $q_1,\ldots,q_n$. Through this, counters $c_{\mathrm{q}},c_{\mathrm{p}}$ are maintained which count the number of accesses to $\mathbf{M}$ (to retrieve a value) and HS (to filter based on a predicate). The labels or predicates associated to each of these subqueries are logged in the vectors $\mathbf{q},\mathbf{p}$. The $r_1,\ldots,r_n$ returned during these calls are part of the leakage. It reveals the "structure" of each query that was made.

Vector $\mathbf{q}$ are the queries made to the multimap primitive and is therefore an input to $\mathcal{L}$. The output of this makes up the multimap leakage (e.g. multimap query equality pattern) and will be returned by $\mathcal{L}^{\mathrm{p}}$. Vector $\mathbf{p}$ is given as input to QP and HF which compute the equality pattern and filtering results on the hashset predicates respectively.

We can now state and prove PpSj's security with respect to $\mathcal{L}^{\mathrm{p}}$.

Theorem 15 *Let $\mathcal{L},\mathcal{S}$ be the leakage algorithm and simulator for* Mme *respectively. Let $\mathcal{L}^{\mathrm{p}},\mathcal{S}^{\mathrm{p}}$ be as defined in Fig. 2.11 and let* F *be the function family used. Then for all adversaries A there exists adversaries $A_{\mathrm{m}},A_{\mathrm{f}}$ such that:*

$$\mathbf{Adv}^{\mathrm{ss}}_{\mathsf{PpSj},\mathcal{L}^{\mathrm{p}},\mathcal{S}^{\mathrm{p}}}(A) \leq \mathbf{Adv}^{\mathrm{ss}}_{\mathsf{Mme},\mathcal{L},\mathcal{S}}(A_{\mathrm{m}}) + (p+1)\cdot \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(A_{\mathrm{f}}).$$

Here, p is the number of distinct predicates used in constructing HS.

Proof. Adversary A_{m} is given in Fig. 2.12. In the same diagram, we see A_1,A_2 which are both PRF adversaries playing $\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}$. We define A_{f} to randomly pick one at run time and use it.

Now we can proceed via a standard hybrid argument. Let $b_{\mathrm{p}},b_{\mathrm{f}},b_{\mathrm{m}}$ be the challenge bits in $\mathrm{G}^{\mathrm{ss}}_{\mathsf{PpSj},\mathcal{L}^{\mathrm{p}},\mathcal{S}^{\mathrm{p}}}$, $\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}$ and $\mathrm{G}^{\mathrm{ss}}_{\mathsf{Mme},\mathcal{L},\mathcal{S}}$ respectively.

From the various advantage definitions, we have that

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{ss}}_{\mathsf{PpSj},\mathcal{L}^{\mathrm{p}},\mathcal{S}^{\mathrm{p}}}(A) &= \Pr[\mathrm{G}^{\mathrm{ss}}_{\mathsf{PpSj},\mathcal{L}^{\mathrm{p}},\mathcal{S}^{\mathrm{p}}}(A)|b_{\mathrm{p}}=1] - \Pr[\mathrm{G}^{\mathrm{ss}}_{\mathsf{PpSj},\mathcal{L}^{\mathrm{p}},\mathcal{S}^{\mathrm{p}}}(A)|b_{\mathrm{p}}=0],\\
\mathbf{Adv}^{\mathrm{ss}}_{\mathsf{Mme},\mathcal{L},\mathcal{S}}(A_{\mathrm{m}}) &= \Pr[\mathrm{G}^{\mathrm{ss}}_{\mathsf{Mme},\mathcal{L},\mathcal{S}}(A_{\mathrm{m}})|b_{\mathrm{m}}=1] - \Pr[\mathrm{G}^{\mathrm{ss}}_{\mathsf{Mme},\mathcal{L},\mathcal{S}}(A_{\mathrm{m}})|b_{\mathrm{m}}=0],\\
\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(A_1) &= \Pr[\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(A_1)|b_{\mathrm{f}}=1] - \Pr[\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(A_1)|b_{\mathrm{f}}=0].
\end{aligned}$$

Notice also that

$$\Pr[\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(A_2)|b_{\mathrm{f}}=0,c=i] = \Pr[\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(A_2)|b_{\mathrm{f}}=1,c=i+1]$$

for $i\in[p-1]$ and

$$\Pr[\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(A_2)|b_{\mathrm{f}}=1,c=j] - \Pr[\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(A_2)|b_{\mathrm{f}}=1,c=j] \leq \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(A_2)$$

for $j \in [p]$. This means that

$$p \cdot \mathbf{Adv}^{\text{prf}}_{\mathsf{F}}(A_2) \geq \Pr[\mathrm{G}^{\text{prf}}_{\mathsf{F}}(A_2) | b_{\text{f}} = 1, c = p] - \Pr[\mathrm{G}^{\text{prf}}_{\mathsf{F}}(A_1) | b_{\text{f}} = 0, c = 1].$$

Notice that A_{m} in $\mathrm{G}^{\text{ss}}_{\mathsf{Mme},\mathcal{L},\mathcal{S}}$ uses the game to simulate multimap encryption and performs the rest itself as it happens in the "real world" of $\mathrm{G}^{\text{ss}}_{\mathsf{PpSj},\mathcal{L}^{\text{p}},\mathcal{S}^{\text{p}}}(A)$. This gives

$$\Pr[\mathrm{G}^{\text{ss}}_{\mathsf{PpSj},\mathcal{L}^{\text{p}},\mathcal{S}^{\text{p}}}(A) | b_{\text{p}} = 1] = \Pr[\mathrm{G}^{\text{ss}}_{\mathsf{Mme},\mathcal{L},\mathcal{S}}(A_{\text{m}}) | b_{\text{m}} = 1].$$

Similarly, A_1 simulates multimap encryption as in the "ideal world" of $\mathrm{G}^{\text{ss}}_{\mathsf{Mme},\mathcal{L},\mathcal{S}}$ and defers the filtering key production to FN which gives us $\Pr[\mathrm{G}^{\text{ss}}_{\mathsf{Mme},\mathcal{L},\mathcal{S}}(A_{\text{m}}) | b_{\text{m}} = 0] = \Pr[\mathrm{G}^{\text{prf}}_{\mathsf{F}}(A_1) | b_{\text{f}} = 1]$. When A_2 plays $\mathrm{G}^{\text{prf}}_{\mathsf{F}}(A_2)$, if $c = p$ then all the K_i will be randomly selected. This means $\Pr[\mathrm{G}^{\text{prf}}_{\mathsf{F}}(A_1) | b_{\text{f}} = 0] = \Pr[\mathrm{G}^{\text{prf}}_{\mathsf{F}}(A_2) | b_{\text{f}} = 1, c = p]$. Over p hybrids, we get to the version where all the $\mathsf{F.Ev}(K_i, \cdot)$ (where K_i is not revealed to the adversary) are simulated with random functions, giving us $\Pr[\mathrm{G}^{\text{prf}}_{\mathsf{F}}(A_1) | b_{\text{f}} = 0, c = 1] = \Pr[\mathrm{G}^{\text{ss}}_{\mathsf{PpSj},\mathcal{L}^{\text{p}},\mathcal{S}^{\text{p}}}(A) | b_{\text{p}} = 0]$ because this selects all of HS elements as $\mathcal{S}^{\text{p}}$ does.

The leakage algorithm $\mathcal{L}^{\text{f}}$ for FpSj is given in Fig. 2.13.

As mentioned in Section 2.4.2, the differences between this and $\mathcal{L}^{\text{p}}$ all stem from their different handling of joins. As depicted in Fig. 2.11, we break down the latter's query-dependent leakage into the recursion structure ($\mathbf{r}$ computed by RS), leakage due to queries to the underlying multimap encryption scheme (lk computed using $\mathcal{L}$), hashset filtering results (SET' computed using HF), hashset query patterns (computed using $\mathsf{QP}(\mathbf{p})$) and the total number of hashset predicates made (c_p). For examples and intuition of each of these forms of leakage, see the examples given in Section 2.4.2.

With $\mathcal{L}^{\text{f}}$, we must compute leakage due to three different types of join queries (leaf, internal or internal intermediate). For leaf joins, the difference in leakage for the two SjDT StI schemes is exactly that of the two JnDT schemes given in Section 2.4. For internal intermediate

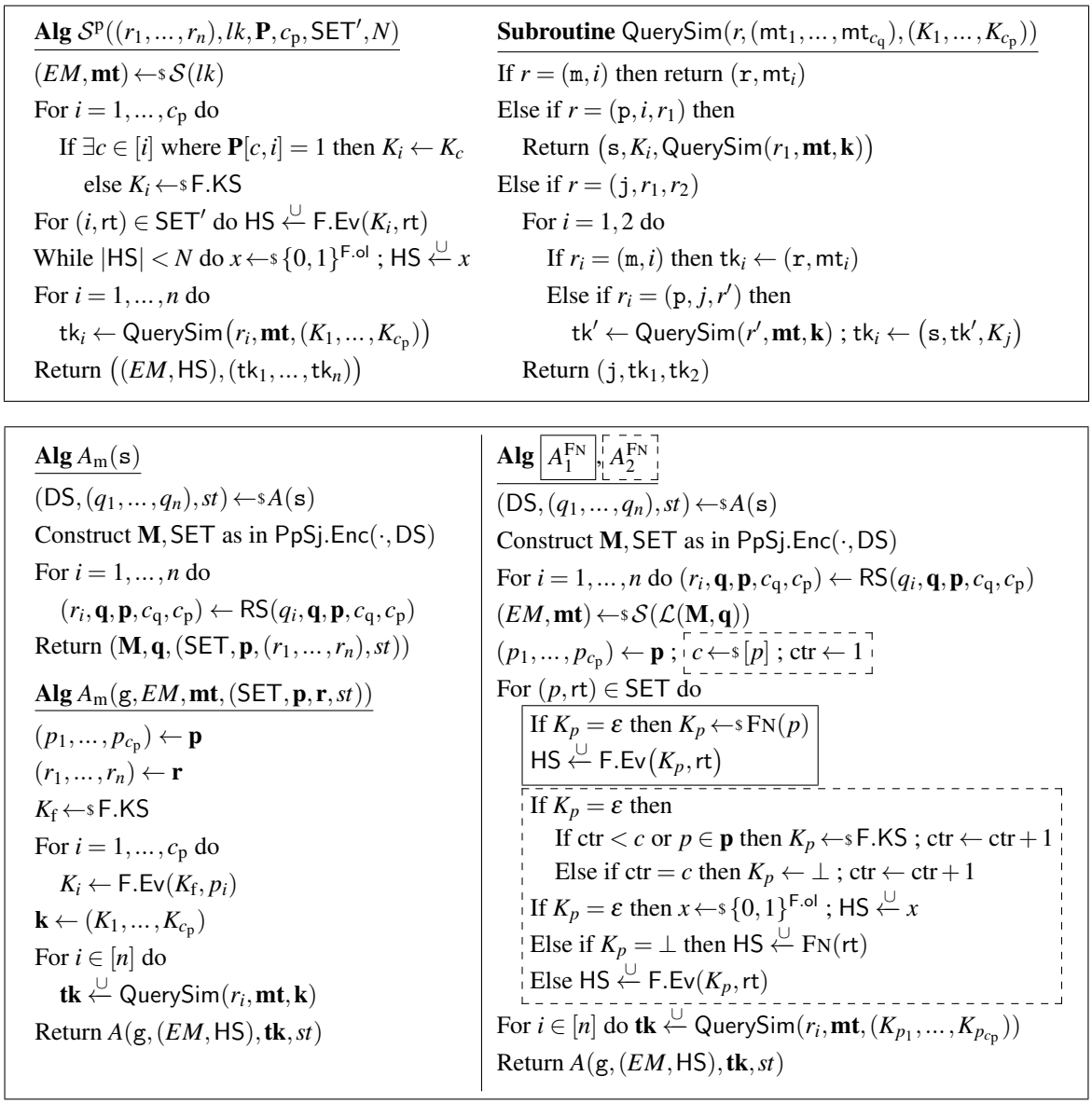

Figure 2.12. Simulator (top) and adversaries (bottom) used in the proof of Theorem 15. In $\mathcal{S}^{\mathrm{p}}$, $\mathcal{S}$ is a simulator for Mme. Note that when A_{f} (from Theorem 15) is run it randomly selects one of A_1, A_2 and runs it.

joins, these are handled entirely using hashset filtering, much like the recursive joins in PpSj. As such, the leakage is comparable (in that we reveal the equality pattern and filtering results of the predicates) with the only subtlety coming from the fact that FpSj associates the join predicate with a pair of row tokens, thereby leakage the equality pattern of the join (but restricted to the rows that have been retrieved by the subqueries).

This leaves the leakage from internal intermediate joins. Recall that the indexing of such joins involved manual additions to the output of $\mathsf{Mme}^{\mathrm{rr}}_{\pi}.\mathsf{Enc}$ (i.e. $\mathbf{D}$). As such, the leakage algorithm must include information to simulate these entries. This includes the final number of values in $\mathbf{D}$ (i.e. M), the length of these values (i.e. ℓ), the query pattern of such joins (i.e. $\mathsf{QP}(\mathbf{j})$), query responses to these (i.e. $\mathbf{I}$) and the number of such joins (i.e. c_{j}).

The security of FpSj Its security is given in Theorem 16 below, which uses the simulator $\mathcal{S}^{\mathrm{f}}$ given in Fig. 2.14.

Theorem 16 *Let $\mathcal{L},\mathcal{S}$ be the leakage algorithm and simulator for $\mathsf{Mme}^{\mathrm{rr}}_{\pi}$ respectively (given in [48]). Let F,SE be the primitives used in $\mathsf{Mme}^{\mathrm{rr}}_{\pi}$ and FpSj's algorithms. Let $\mathcal{L}^{\mathrm{f}},\mathcal{S}^{\mathrm{f}}$ be as defined in Fig. 2.13 and Fig. 2.14 respectively. Then for all adversaries A there exists adversaries $A_{\mathrm{f}},A_{\mathrm{s}}$ such that:*

$$\mathbf{Adv}^{\mathrm{ss}}_{\mathsf{PpSj},\mathcal{L}^{\mathrm{p}},\mathcal{S}^{\mathrm{p}}}(A) \leq (m+m_1)\mathbf{Adv}^{\mathrm{ind\$}}_{\mathsf{SE}}(A_{\mathrm{s}}) + (m+m_1+p+1).$$

Here, m,m_1 are the number of labels in $\mathbf{M},\mathbf{M}_1$ respectively and p is the number of distinct predicates used in constructing HS.

Proof. This proof is quite similar to Theorem 15, except that we can now reduce security straight to SE,F because we assumed the use of the $\mathsf{Mme}^{\mathrm{rr}}_{\pi}$ multimap encryption scheme. We therefore omit a full description of the adversaries and proof except to say that the multiplicative factors in the bound come from the number of SE and F keys that are used in $\mathsf{FpSj}.\mathsf{Enc}$ (including those which are an output of $\mathsf{F}.\mathsf{Ev}$).

Alg $\mathcal{L}^{\mathrm{f}}\big(\mathsf{DS},(q_1,\ldots,q_n)\big)$

Construct $\mathbf{M},\mathbf{M}_1,\mathsf{SET}$ as in $\mathsf{FpSj.Enc}(\cdot,\mathsf{DS})$
For $i=1,\ldots,n$ do
  $(r_i,\mathbf{q},\mathbf{p},\mathbf{j},c_{\mathrm{q}},c_{\mathrm{p}},c_{\mathrm{j}}) \leftarrow \mathsf{RS}(q_i,\mathbf{q},\mathbf{p},\mathbf{j},c_{\mathrm{q}},c_{\mathrm{p}},c_{\mathrm{j}})$
$\mathbf{r} \leftarrow (r_1,\ldots,r_n)$; $lk \leftarrow\!\!\$\ \mathcal{L}(\mathbf{M},\mathbf{q})$; $S \leftarrow \bigcup_{q\in\mathbf{q}} \mathbf{M}[q]$
While $S \neq S'$ do $S \leftarrow S'$; $(\mathsf{SET},S',\mathbf{I}) \leftarrow \mathsf{IJ}(\mathbf{j},S',\mathbf{I})$
$\mathsf{SET}' \leftarrow \mathsf{HF}(\mathbf{p},S,\mathsf{SET})$
Define M : # of vals in $\mathbf{M}$ and $\mathbf{M}_1$
Define ℓ : max. length val in $\mathbf{M},\mathbf{M}_1$
Return $(\mathbf{r},lk,\mathsf{QP}(\mathbf{p}),c_{\mathrm{p}},\mathsf{SET}',|\mathsf{SET}'|,\mathbf{I},M,\ell,\mathsf{QP}(\mathbf{j}),c_{\mathrm{j}})$

Subroutine $\mathsf{IJ}\big((j_1,\ldots,j_n),S,\mathbf{I}\big)$

For $i\in[n]$ $\mathsf{rt}\in S$ do
  $(\mathbf{t}_1,\mathbf{t}_2,k) \leftarrow j_i$
  If $\mathbf{M}_1[(\mathbf{t}_1,\mathbf{t}_2,\mathsf{rt},i)] \neq \perp$ then
    $S' \leftarrow S' \cup \mathbf{M}_1[(\mathbf{t}_1,\mathbf{t}_2,\mathsf{rt},i)]$; $\mathbf{I}[(\mathsf{rt},i)] \leftarrow \mathbf{M}_1[(\mathbf{t}_1,\mathbf{t}_2,\mathsf{rt},i)]$
Return $(S',\mathbf{I})$

Subroutine $\mathsf{HF}(\mathbf{p},S,\mathsf{SET})$

$(p_1,\ldots,p_n) \leftarrow \mathbf{p}$
For all $i\in[n]$
  If $p_i = (\mathtt{iij},\mathbf{t}_1,\mathbf{t}_2)$ then
    For $\mathbf{rt}\in S\times S$ do
      If $(p_i,\mathbf{rt})\in\mathsf{SET}$ then
        $\mathsf{SET}' \xleftarrow{\cup} (i,\mathbf{rt})$
  Else
    For $\mathsf{rt}\in S$ do
      If $(p_i,\mathsf{rt})\in\mathsf{SET}$ then
        $\mathsf{SET}' \xleftarrow{\cup} (i,\mathsf{rt})$
Return SET'

Subroutine $\mathsf{QP}\big((t_1,\ldots,t_n)\big)$

For all $i,j\in[n]$ if $t_i = t_j$ then
  $\mathbf{T}[i,j] \leftarrow 1$ else $\mathbf{T}[i,j] \leftarrow 0$
Return $\mathbf{T}$

Subroutine $\mathsf{RS}(q,\mathbf{q},\mathbf{p},\mathbf{j},c_{\mathrm{q}},c_{\mathrm{p}},c_{\mathrm{j}})$

If $q=(\mathtt{r},id)$ then $\mathbf{q} \xleftarrow{\cup} (\mathtt{r},id)$; $r \leftarrow (\mathtt{m},c_{\mathrm{q}})$; $c_{\mathrm{q}} \leftarrow c_{\mathrm{q}}+1$
Else if $q=(\mathtt{s},at,x,(\mathtt{r},id))$ then $\mathbf{q} \xleftarrow{\cup} (\mathtt{s},at,x)$; $r \leftarrow (\mathtt{m},c_{\mathrm{q}})$; $c_{\mathrm{q}} \leftarrow c_{\mathrm{q}}+1$
Else if $q=(\mathtt{s},at,x,q_1)$ then
  $(r_1,\mathbf{q},\mathbf{p},\mathbf{j},c_{\mathrm{q}},c_{\mathrm{p}},c_{\mathrm{j}}) \leftarrow \mathsf{RS}(q_1,\mathbf{q},\mathbf{p},\mathbf{j},c_{\mathrm{q}},c_{\mathrm{p}},c_{\mathrm{j}})$; $\mathbf{p} \xleftarrow{\cup} (\mathtt{s},at,x)$; $r \leftarrow (\mathtt{s},c_{\mathrm{p}},r_1)$; $c_{\mathrm{p}} \leftarrow c_{\mathrm{p}}+1$
Else if $q=(\mathbf{t}_1,\mathbf{t}_2,(\mathtt{r},id_1),(\mathtt{r},id_2))$ then $\mathbf{q} \xleftarrow{\cup} (\mathtt{j},\mathbf{t}_1,\mathbf{t}_2)$; $r \leftarrow (\mathtt{m},c_{\mathrm{q}})$; $c_{\mathrm{q}} \leftarrow c_{\mathrm{q}}+1$
Else if $q=(\mathbf{t}_1,\mathbf{t}_2,q_1,(\mathtt{r},id))$ or $q=(\mathbf{t}_1,\mathbf{t}_2,(\mathtt{r},id),q_1)$ then
  $(r_1,\mathbf{q},\mathbf{p},\mathbf{j},c_{\mathrm{q}},c_{\mathrm{p}},c_{\mathrm{j}}) \leftarrow \mathsf{RS}(q_1,\mathbf{q},\mathbf{p},\mathbf{j},c_{\mathrm{q}},c_{\mathrm{p}},c_{\mathrm{j}})$; If $q=(\mathbf{t}_1,\mathbf{t}_2,q_1,(\mathtt{r},id))$ then $i=1$ else $i=2$
  $\mathbf{j} \xleftarrow{\cup} (\mathbf{t}_1,\mathbf{t}_2,i)$; $r \leftarrow (\mathtt{ij},c_{\mathrm{j}},i,r_1)$; $c_{\mathrm{j}} \leftarrow c_{\mathrm{j}}+1$
Else if $q=(\mathbf{t}_1,\mathbf{t}_2,q_1,q_2)$ then
  For $i=1,2$ do $(r_i,\mathbf{q},\mathbf{p},\mathbf{j},c_{\mathrm{q}},c_{\mathrm{p}},c_{\mathrm{j}}) \leftarrow \mathsf{RS}(q_i,\mathbf{q},\mathbf{p},\mathbf{j},c_{\mathrm{q}},c_{\mathrm{p}},c_{\mathrm{j}})$
  $\mathbf{p} \xleftarrow{\cup} (\mathtt{iij},\mathbf{t}_1,\mathbf{t}_2)$; $r \leftarrow (\mathtt{iij},c_{\mathrm{p}},r_1,r_2)$; $c_{\mathrm{p}} \leftarrow c_{\mathrm{p}}+1$
Return $(r,\mathbf{q},\mathbf{p},\mathbf{j},c_{\mathrm{q}},c_{\mathrm{p}})$

Figure 2.13. Leakage profile for FpSj. Here, $\mathcal{L}$ is the leakage algorithm for $\mathsf{Mme}^{\mathrm{rr}}_{\pi}$ and subroutines $\mathsf{IJ},\mathsf{HF},\mathsf{QP},\mathsf{RS}$ compute the leakage associated to internal joins, hashset filtering, hashset query patterns and query recursion structures.

Alg $\mathcal{S}^{\mathrm{f}}(\mathbf{r}, lk, \mathbf{P}, c_{\mathrm{p}}, \mathsf{SET}', N, \mathbf{I}, M, \ell, \mathbf{J}, c_{\mathrm{j}})$

$(r_1, \ldots, r_n) \leftarrow \mathbf{r}$; $(\mathbf{D}, \mathbf{mt}) \leftarrow\!\!\$\ \mathcal{S}(lk)$
For $i = 1, \ldots, c_{\mathrm{p}}$ do
  If $\exists c \in [i]$ where $\mathbf{P}[c,i] = 1$ then $K_i \leftarrow K_c$ else $K_i \leftarrow\!\!\$\ \mathsf{F.KS}$
For $(i,x) \in \mathsf{SET}'$ do $\mathsf{HS} \xleftarrow{\cup} \mathsf{F.Ev}(K_i, x)$
While $|\mathsf{HS}| < N$ do $x \leftarrow\!\!\$\ \{0,1\}^{\mathsf{F.ol}}$; $\mathsf{HS} \xleftarrow{\cup} x$
For $(\mathsf{rt}, i) \in \mathbf{I}.\mathsf{Lbls}$ do
  If $\exists c \in [i]$ where $\mathbf{J}[c,i] = 1$ then $K'_i \leftarrow K'_c$ else $K'_i \leftarrow\!\!\$\ \mathsf{F.KS}$
  $\{\mathsf{rt}'_1, \ldots, \mathsf{rt}'_n\} \leftarrow \mathbf{I}[(\mathsf{rt}, i)]$
  For $k = 0, 1$ do $K''_k \leftarrow \mathsf{F.Ev}(K'_i, \mathsf{rt} \| k)$
  For $k \in [n]$ do
    Pad rt'_k to length ℓ
    $\mathbf{D}[\mathsf{F.Ev}(K''_0, k)] \leftarrow \mathsf{SE.Enc}(K''_1, \mathsf{rt}'_k)$
While $|\mathbf{D}.\mathsf{Lbls}| < M$ do $x \leftarrow\!\!\$\ \{0,1\}^{\mathsf{F.ol}}$; $\mathbf{D}[x] \leftarrow\!\!\$\ \{0,1\}^{\ell}$
For $i = 1, \ldots, n$ do
  $\mathsf{tk}_i \leftarrow \mathsf{QuerySim}(r_i, \mathbf{mt}, (K_1, \ldots, K_{c_{\mathrm{p}}}), (K'_1, \ldots, K'_{c_{\mathrm{j}}}))$
Return $\big((\mathbf{D}, \mathsf{HS}), (\mathsf{tk}_1, \ldots, \mathsf{tk}_n)\big)$

Subroutine $\mathsf{QuerySim}(r, \mathbf{mt}, \mathbf{k}, \mathbf{k}')$

$(\mathsf{mt}_1, \ldots, \mathsf{mt}_{c_{\mathrm{q}}}) \leftarrow \mathbf{mt}$
$(K_1, \ldots, K_{c_{\mathrm{p}}}) \leftarrow \mathbf{k}$; $(K'_1, \ldots, K'_{c_{\mathrm{j}}}) \leftarrow \mathbf{k}'$
If $r = (\mathtt{m}, i)$ then return $(\mathtt{r}, \mathsf{mt}_i)$
Else if $r = (\mathtt{p}, i, r_1)$ then
  $\mathsf{tk} \leftarrow \mathsf{QuerySim}(r_1, \mathbf{mt}, \mathbf{k}, \mathbf{k}')$
  Return $(\mathtt{s}, K_i, \mathsf{tk})$
Else if $r = (\mathtt{ij}, i, j, r_1)$ then
  $\mathsf{tk} \leftarrow \mathsf{QuerySim}(r_1, \mathbf{mt}, \mathbf{k}, \mathbf{k}')$
  Return $(\mathtt{ij}, K'_i, j, \mathsf{tk})$
Else if $r = (\mathtt{iij}, i, r_1, r_2)$ then
  For $j = 1, 2$ do
    $\mathsf{tk}_j \leftarrow \mathsf{QuerySim}(r_j, \mathbf{mt}, \mathbf{k}, \mathbf{k}')$
  Return $(\mathtt{iij}, K_i, \mathsf{tk}_1, \mathsf{tk}_2)$

Figure 2.14. Simulator used in the proof of Theorem 16 where $\mathcal{S}$ is a simulator for $\mathsf{Mme}^{\mathrm{rr}}_{\pi}$.

Efficiency drawback of PpSj**.**

Comparing the bandwidth of PpSj, FpSj is also not clear cut: On non-recursive queries, PpSj will perform equal or better than FpSj but on recursive queries the converse is sometimes true.

Consider the query $q = (\mathtt{s}, at_3, \mathtt{CS}, (\mathtt{j}, at_1, at_2, (\mathtt{r}, id_1), (\mathtt{r}, id_2)))$ in our toy example. With FpSj, the server returns pointers to the two rows that feature in the output relation (i.e. those with coordinates $(id_1, \mathtt{cc}), (id_2, \mathtt{33})$) but PpSj returns four (i.e. with coordinates $(id_1, \mathtt{aa}), (id_1, \mathtt{bb})$, $(id_1, \mathtt{cc}), (id_2, \mathtt{33})$) because without the equality pattern over the join columns and it cannot filter out the first and second rows of R_1.

More generally, this overhead may occur anytime that a recursive query (involving at least one join) is made and grows with query complexity. Depending on the data and query workload, this overhead ranges from negligible to quite substantial, something we explore further in Section 2.6.

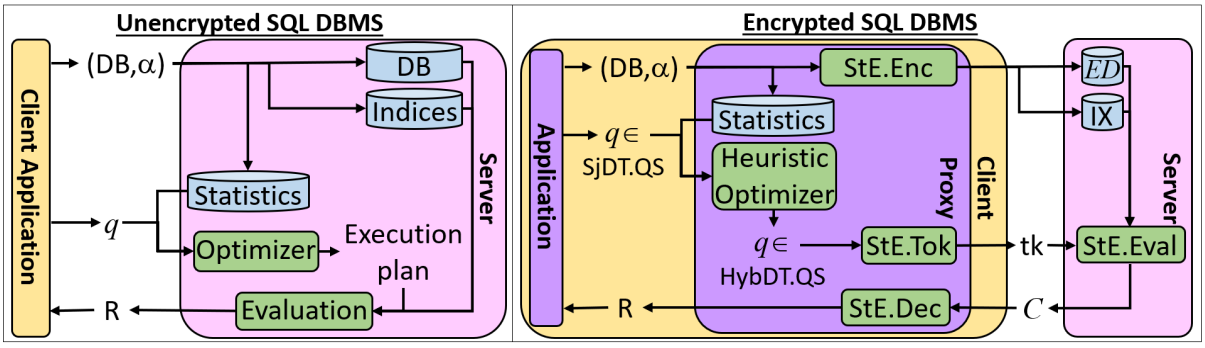

Figure 2.15. Data/ query processing in unencrypted SQL databases (left) and the analogous processes using **SqlStE** with hybrid indexing (right).

2.5 Hybrid indexing

We showed that the choice between FP and PP indexing depends heavily on query load. This motivates our hybrid StI scheme that postpones this decision till query time. We first cover the technical details of supporting both indexing techniques, then give a heuristic for the client to choose between them.

Hybrid data processing.

We give a new ADT where each join in a query is annotated with the desired indexing technique, HybDT. This ADT is equivalent to SjDT except that its join queries take the form $(op, \mathbf{t}_1, \mathbf{t}_2, q_1, q_2)$ where $op \in \{\texttt{fp}, \texttt{pp}\}$. When evaluating HybDT.Spec, these are both functionally equivalent to the analogous SjDT join query's $(\texttt{j}, \mathbf{t}_1, \mathbf{t}_2, q_1, q_2)$.

The hybrid system we envision makes the same assumption as in (unencrypted) SQL DBMSes – that client queries are unoptimized and have no canonical form – and therefore mirrors its data flow as depicted in Fig. 2.15. It also borrows its architecture (i.e. use of a client-side proxy) from existing encrypted SQL solutions [116, 130]. The client's SjDT query is annotated using a heuristic optimizer to get a HybDT query. This latter query is then tokenized, evaluated and decrypted using hybrid indexing scheme HybStI in $\mathsf{StE} = \textbf{SqlStE}[\mathsf{HybStI}, \mathsf{SE}, \mathsf{F}]$.

As best we know, no existing work has looked into query optimization in StE schemes. We believe this area to be of independent interest because unlike encrypted systems where

optimization runs on the server (with full access to the data) and is solely interested in maximizing efficiency, optimization in encrypted SQL DBMSes should be done (at least partially) by the proxy with only precomputed statistics about the data and may additionally seek to minimize leakage. We initiate this study with our heuristic below.

HybStI **details.**

This StI merges FpSj, PpSj by essentially storing both kinds of indexes on the server. More specifically, HybStI.Enc will merge the multimaps and hashsets generated by PpSj, FpSj (avoiding repetition where possible) so that it can take join tokens of either form. When a HybDT join query is made, the client indicates which index to use in its query with op.

We believe the intuition for how HybStI works is straightforward. The full pseudocode of HybStI's algorithms is given in Fig. 2.16. Note that HybFinalize is a subroutine recursively called by HybStI.Fin to perform client-side (i.e. PP) joins.

The only subtlety comes when a query contains both FP and PP joins. Notice that pointer tuples in this case will contain more than one P_i (unlike FpSj) and the tuples in at least one P_i will contain more than one rt (unlike PpSj). As such, after the client performs the PP joins in HybStI.Fin some column reordering may be necessary. This is done within HybStI.Fin which can use scma, q to compute the desired order of attributes in the output relation.

HybStI **leakage.**

We will describe HybStI's leakage profile in comparison to that of PpSj and FpSj. The metadata leakage is comparable, with each size (multimap or hashset) being the sum of respective FpSj and PpSj sizes. The recursion structure leakage is technically higher but only because we leak the join annotations that weren't present in the other two schemes.

For the same reason that PpSj and FpSj's query-dependent leakages were not directly comparable, they also cannot be compared with that of HybStI. However, like we did in Section 2.4.2, we can upper bound HybStI's query-dependent leakage on $q_1, \ldots, q_n \in$ HybDT.QS

Alg $\mathsf{HybStI.Enc}\big(K_{\mathrm{m}},(\mathsf{DB},\alpha,\mathbf{T})\big)$

For all $(id,\mathsf{R})\in\mathsf{DB}$ and $\mathbf{r}\in\mathsf{R.T}$ do
  $\mathsf{rt}\leftarrow\mathbf{T}[(id,\mathbf{r}[\mathsf{uk}(id)])]$; $\mathbf{M}[(\mathtt{r},id)]\xleftarrow{\cup}(\mathsf{rt})$
  For $at\in\mathsf{R.Ats}$ where $at\neq\mathsf{uk}(id)$ do
    $\mathbf{M}[(\mathtt{s},at,\mathbf{r}[at])]\xleftarrow{\cup}(\mathsf{rt})$; $\mathsf{SET}\xleftarrow{\cup}\big((\mathtt{s},at,\mathbf{r}[at]),\mathsf{rt}\big)$
For $(\mathbf{t}_1,\mathbf{t}_2)\in\alpha$ do
  $id_1\leftarrow\mathsf{getID}(\mathbf{t}_1)$; $id_2\leftarrow\mathsf{getID}(\mathbf{t}_2)$
  For $\mathbf{r}\in\big(\mathsf{DB}[id_1]\bowtie_{\mathbf{t}_1,\mathbf{t}_2}\mathsf{DB}[id_2]\big).\mathsf{T}$ do
    For $i=1,2$ do
      $\mathsf{rt}_i\leftarrow\mathbf{T}[(id_i,\mathbf{r}[\mathsf{uk}(id_i)])]$
      $\mathbf{M}[(\mathtt{pp},\mathbf{t}_1,\mathbf{t}_2,i)]\xleftarrow{\cup}(\mathsf{rt}_i)$
      $\mathsf{SET}\xleftarrow{\cup}\big((\mathtt{pp},\mathbf{t}_1,\mathbf{t}_2,i),\mathsf{rt}_i\big)$
    $\mathbf{M}[(\mathtt{fp},\mathbf{t}_1,\mathbf{t}_2)]\xleftarrow{\cup}(\mathsf{rt}_1,\mathsf{rt}_2)$
    $\mathbf{M}_1[(\mathbf{t}_1,\mathbf{t}_2,\mathsf{rt}_1,1)]\xleftarrow{\cup}\mathsf{rt}_2$; $\mathbf{M}_1[(\mathbf{t}_1,\mathbf{t}_2,\mathsf{rt}_2,2)]\xleftarrow{\cup}\mathsf{rt}_1$
    $\mathsf{SET}\xleftarrow{\cup}\big((\mathtt{fp},\mathbf{t}_1,\mathbf{t}_2),(\mathsf{rt}_1,\mathsf{rt}_2)\big)$
$(K_{\mathrm{m}},\mathbf{D})\leftarrow\!\!\$\,\mathsf{Mme}^{\mathrm{rr}}_{\pi}.\mathsf{Enc}(K_{\mathrm{m}},\mathbf{M})$
For $(\mathbf{t}_1,\mathbf{t}_2,\mathsf{rt},i)\in\mathbf{M}_1.\mathsf{Lbls}$ do
  For $j=0,1$ do
    $K_j\leftarrow\mathsf{F.Ev}(\mathsf{F.Ev}(K_{\mathrm{m}},(\mathtt{ij},\mathbf{t}_1,\mathbf{t}_2,i)),\mathsf{rt}\|j)$
  $\{\mathsf{rt}_1,\ldots,\mathsf{rt}_n\}\leftarrow\mathbf{M}_1[(\mathbf{t}_1,\mathbf{t}_2,\mathsf{rt},i)]$
  For $k\in[n]$ do
    Pad rt_k to $\mathbf{M}$'s max. value length
    $\mathbf{D}[\mathsf{F.Ev}(K_0,k)]\leftarrow\!\!\$\,\mathsf{SE.Enc}(K_1,\mathsf{rt}_k)$
$K_{\mathrm{f}}\leftarrow\!\!\$\,\mathsf{F.KS}$; $\mathsf{HS}\leftarrow\mathsf{HsEnc}(K_{\mathrm{f}},\mathsf{SET})$
Return $\big((\mathsf{Schema}(\mathsf{DB}),K_{\mathrm{m}},K_{\mathrm{f}}),(\mathbf{D},\mathsf{HS})\big)$

Alg $\mathsf{HybStI.Tok}(K_{\mathrm{i}},q)$

$(\mathsf{scma},K_{\mathrm{m}},K_{\mathrm{f}})\leftarrow K_{\mathrm{i}}$
If $q=(\mathtt{r},id)$ then return $\big(\mathtt{r},\mathsf{Mme}^{\mathrm{rr}}_{\pi}.\mathsf{Tok}(K_{\mathrm{m}},(\mathtt{r},id))\big)$
Else if $q=(\mathtt{s},at,x,(\mathtt{r},id))$ then
  Return $\big(\mathtt{r},\mathsf{Mme}^{\mathrm{rr}}_{\pi}.\mathsf{Tok}(K_{\mathrm{m}},(\mathtt{s},at,x))\big)$
Else if $q=(\mathtt{s},at,x,q_1)$ then
  Return $\big(\mathtt{s},\mathsf{F.Ev}(K_{\mathrm{f}},(\mathtt{s},at,x)),\mathsf{HybStI.Tok}(K_{\mathrm{i}},q_1)\big)$
Else if $q=(\mathtt{pp},\mathbf{t}_1,\mathbf{t}_2,q_1,q_2)$ then
  For $i=1,2$ do
    If $q_i=(\mathtt{r},id_i)$ then
      $\mathsf{tk}_i\leftarrow\!\!\$\,\big(\mathtt{r},\mathsf{Mme}^{\mathrm{rr}}_{\pi}.\mathsf{Tok}(K_{\mathrm{m}},(\mathtt{pp},\mathbf{t}_1,\mathbf{t}_2,i))\big)$
    Else
      $\mathsf{tk}'\leftarrow\!\!\$\,\mathsf{HybStI.Tok}(K_{\mathrm{i}},q_i)$
      $\mathsf{tk}_i\leftarrow\big(\mathtt{s},\mathsf{F.Ev}(K_{\mathrm{f}},(\mathtt{pp},\mathbf{t}_1,\mathbf{t}_2,i)),\mathsf{tk}'\big)$
  Return $(\mathtt{pp},\mathsf{tk}_1,\mathsf{tk}_2)$
Else if $q=(\mathtt{fp},\mathbf{t}_1,\mathbf{t}_2,(\mathtt{r},id_1),(\mathtt{r},id_2))$ then
  Return $\big(\mathtt{r},\mathsf{Mme}^{\mathrm{rr}}_{\pi}.\mathsf{Tok}(K_{\mathrm{m}},(\mathtt{fp},\mathbf{t}_1,\mathbf{t}_2))\big)$
Else if $q=(\mathtt{fp},\mathbf{t}_1,\mathbf{t}_2,q_1,(\mathtt{r},id))$ then
  Return $(\mathtt{ij},\mathsf{F.Ev}(K_{\mathrm{m}},(\mathtt{ij},\mathbf{t}_1,\mathbf{t}_2,1)),\mathsf{HybStI.Tok}(K_{\mathrm{i}},q_1))$
Else if $q=(\mathtt{fp},\mathbf{t}_1,\mathbf{t}_2,(\mathtt{r},id),q_1)$ then
  Return $(\mathtt{ij},\mathsf{F.Ev}(K_{\mathrm{m}},(\mathtt{ij},\mathbf{t}_1,\mathbf{t}_2,2)),\mathsf{HybStI.Tok}(K_{\mathrm{i}},q_1))$
Else if $q=(\mathtt{fp},\mathbf{t}_1,\mathbf{t}_2,q_1,q_2)$ then
  For $i=1,2$ do $\mathsf{tk}_i\leftarrow\!\!\$\,\mathsf{HybStI.Tok}(K_{\mathrm{i}},q_i)$
  Return $\big(\mathtt{iij},\mathsf{F.Ev}(K_{\mathrm{f}},(\mathtt{fp},\mathbf{t}_1,\mathbf{t}_2)),\mathsf{tk}_1,\mathsf{tk}_2\big)$

Alg $\mathsf{HybStI.Eval}(\mathsf{tk},\mathsf{IX})$

$(\mathbf{D},\mathsf{HS})\leftarrow\mathsf{IX}$
If $\mathsf{tk}=(\mathtt{r},\mathsf{tk}_1)$ then return $(\mathsf{Mme}^{\mathrm{rr}}_{\pi}.\mathsf{Eval}(\mathsf{tk},\mathsf{IX}))$
Else If $\mathsf{tk}=(\mathtt{s},K,\mathsf{tk}_1)$ then
  Return $\mathsf{HsFilter}\big(K,\mathsf{HybStI.Eval}(\mathsf{tk}_1,\mathsf{IX}),\mathsf{HS}\big)$
Else if $\mathsf{tk}=(\mathtt{pp},\mathsf{tk}_1,\mathsf{tk}_2)$
  Return $\mathsf{HybStI.Eval}(\mathsf{tk}_1,\mathsf{IX})\|\mathsf{HybStI.Eval}(\mathsf{tk}_2,\mathsf{IX})$
Else if $\mathsf{tk}=(\mathtt{ij},K,\mathsf{tk}_1)$ then
  $(P_1,\ldots,P_n)\leftarrow\mathsf{HybStI.Eval}(\mathsf{tk}_1,\mathsf{IX})$
  Define $j:\exists\mathsf{rt}\in\mathbf{rt}\in P_j$
    where $\mathbf{D}[\mathsf{F.Ev}(\mathsf{F.Ev}(K,\mathsf{rt}\|0),0)]\neq\perp$
  For $\mathsf{rt}\in\mathbf{rt}\in P_j$ do
    For $i=1,2$ do $K_i\leftarrow\mathsf{F.Ev}(K,\mathsf{rt}\|i)$
    While $\mathbf{D}[\mathsf{F.Ev}(K_0,c_{\mathsf{rt}})]\neq\perp$ do
      $\mathbf{rt}\xleftarrow{\cup}\mathsf{SE.Dec}(K_1,\mathbf{D}[\mathsf{F.Ev}(K_0,c_{\mathsf{rt}})])$
      $P'\xleftarrow{\cup}\mathbf{rt}$; $c_{\mathsf{rt}}\leftarrow c_{\mathsf{rt}}+1$
  Return $P\setminus\{P_j\}\|(P')$
Else if $\mathsf{tk}=(\mathtt{iij},K,\mathsf{tk}_1,\mathsf{tk}_2)$ then
  For $i=1,2$ do
    $(P^i_1,\ldots,P^i_{n_i})\leftarrow\mathsf{HybStI.Eval}(\mathsf{tk}_i,(\mathbf{D},\mathsf{HS}))$
  Define $j_1,j_2:\exists\mathsf{rt}_i\in\mathbf{rt}_i\in P^i_{j_i}$
    where $\mathsf{F.Ev}(K,(\mathsf{rt}_1,\mathsf{rt}_2))\in\mathsf{HS}$
  For $\mathsf{rt}_1\in\mathbf{rt}_1\in P^1_{j_1}$ and $\mathsf{rt}_2\in\mathbf{rt}_2\in P^2_{j_2}$ do
    If $\mathsf{F.Ev}(K,(\mathsf{rt}_1,\mathsf{rt}_2))\in\mathsf{HS}$ then $P'\xleftarrow{\cup}\mathbf{rt}_1\|\mathbf{rt}_2$
  If $P'\neq\emptyset$ then return $P_1\setminus\{P^1_{j_1}\}\|P_2\setminus\{P^2_{j_2}\}\|(P')$

Alg $\mathsf{HybStI.Fin}\big((\mathsf{scma},K_{\mathrm{m}},K_{\mathrm{f}}),q,(M_1,\ldots,M_n)\big)$

Using scma and q, compute $\mathbf{at_1},\ldots,\mathbf{at_n},\mathbf{at}$, the attributes in $M_1,\ldots,M_n,\mathsf{HybDT.Spec}(q,\mathsf{DS})$ respectively
For $i\in[n]$ do
  $\mathsf{R}_i\leftarrow\mathsf{NewRltn}(\mathbf{at_i})$
  $\mathsf{R}_i.\mathsf{T}\leftarrow\{m_1\|\ldots\|m_{n'}:(m_1,\ldots,m_{n'})\in M_i\}$
$\mathsf{R}\leftarrow\mathsf{HybFinalize}\big(q,(\mathsf{R}_1,\ldots,\mathsf{R}_n)\big)$
If $\mathsf{R.Ats}\neq\mathbf{at}$ then reorder attributes in R accordingly
Return R

Subroutine $\mathsf{HybFinalize}(q,\mathbf{R})$

If $q=(\mathtt{r},id)$ then $(\mathsf{R})\leftarrow\mathbf{R}$; Return R
Else if $q=(\mathtt{s},at,x,q_1)$ then return $\mathsf{HybFinalize}(q_1,\mathbf{R})$
Else if $q=(\mathtt{fp},\mathbf{t}_1,\mathbf{t}_2,q_1,q_2)$ then
  $(\mathsf{R}_1,\ldots,\mathsf{R}_n)\leftarrow\mathbf{R}$
  Define $j:\mathbf{t}_1\cup\mathbf{t}_2\subseteq\mathsf{R}_j.\mathsf{Ats}$
  Partition $\mathbf{R}\setminus\{\mathsf{R}_j\}$ into $\mathbf{R}_1,\mathbf{R}_2$ where $\mathbf{R}_i$ contains all the attributes in $\mathsf{HybDT}(q_i,\mathsf{DS})$
  $\mathsf{R}\leftarrow\mathsf{HybFinalize}(q_1,\mathbf{R}_1\|(\mathsf{R}_j))$
  $\mathsf{R}\leftarrow\mathsf{HybFinalize}(q_2,\mathbf{R}_2\|(\mathsf{R}))$
Else if $q=(\mathtt{pp},\mathbf{t}_1,\mathbf{t}_2,q_1,q_2)$ then
  Partition $\mathbf{R}$ into $\mathbf{R}_1,\mathbf{R}_2$ where $\mathbf{R}_i$ contains all attributes in $\mathsf{HybDT}(q_i,\mathsf{DS})$
  Return $\mathsf{HybFinalize}(q_1,\mathbf{R}_1)\bowtie_{\mathbf{t}_1,\mathbf{t}_2}\mathsf{HybFinalize}(q_2,\mathbf{R}_2)$

Figure 2.16. Algorithms for HybStI, the StI scheme for HybDT using hybrid indexing. HybFinalize is a recursively called subroutine used in HybStI.Fin.

with that of $q'_1, \ldots, q'_m$, the minimal set of non-recursive queries in HybDT.QS (with consistent join annotation) with which the server can still compute its output on $q_1, \ldots, q_n$. This leakage is no better than the analogous bound in PpSj and no worse than that of FpSj, this confirms the intuition that hybrid indexing achieves an *intermediate level of query-dependent leakage* compared to solely using FP or PP indexing.

For completeness, the leakage profile of HybStl is described via pseudocode in Fig. 2.17. This leakage algorithm merges our two previous ones (i.e. $\mathcal{L}^{\mathrm{f}}$ and $\mathcal{L}^{\mathrm{p}}$) in the straightforward way. In particular, the only difference between $\mathcal{L}^{\mathrm{f}}$ (Fig. 2.13) and $\mathcal{L}^{\mathrm{h}}$ is the recursion structure of partially precomputed joins which are handled in the style of $\mathcal{L}^{\mathrm{p}}$ (Fig. 2.11).

The proof of HybStl's security (with respect to $\mathcal{L}^{\mathrm{h}}$) is also very similar to the result for Theorem 16. As such, we state the security bound below but omit the proof for brevity.

Theorem 17 *Let $\mathcal{L}$ be the leakage algorithm and simulator for $\mathsf{Mme}^{\mathrm{rr}}_{\pi}$ (given in [48]) and $\mathcal{L}^{\mathrm{h}}$ be as defined in Fig. 2.17. Let F, SE be the primitives used in $\mathsf{Mme}^{\mathrm{rr}}_{\pi}$ and HybStl's algorithms. Then for all adversaries A there exists $A_{\mathrm{f}}, A_{\mathrm{s}}, \mathcal{S}^{\mathrm{h}}$ such that:*

$$\mathbf{Adv}^{\mathrm{ss}}_{\mathsf{HybStl},\mathcal{L}^{\mathrm{h}},\mathcal{S}^{\mathrm{h}}}(A) \leq (m+m_1)\mathbf{Adv}^{\mathrm{ind\$}}_{\mathsf{SE}}(A_{\mathrm{s}}) + (m+m_1+p+1).$$

Here, m, m_1 are the number of labels in $\mathbf{M}, \mathbf{M}_1$ respectively and p is the number of distinct predicates used in constructing HS.

Leakage-aware query planning.

The join annotation selected by our query planning heuristic will minimize leakage without exceeding a predetermined bandwidth limit. More specifically, suppose the user supplies a query $q \in \mathsf{SjDT}$ with J joins and a bandwidth limit L indicating the maximum number of rows from ED that can be returned in the ciphertext tuple. We estimate the bandwidth of all possible HybDT queries, then select an annotation by:

1. Eliminating options which exceed L rows. If none remain, return $\perp$.

Alg $\mathcal{L}^{\mathrm{h}}\big(\mathsf{DS},(q_1,\ldots,q_n)\big)$

Construct $\mathbf{M},\mathbf{M}_1,\mathsf{SET}$ as in $\mathsf{FpSj.Enc}(\cdot,\mathsf{DS})$
For $i=1,\ldots,n$ do $(r_i,\mathbf{q},\mathbf{p},\mathbf{j},c_{\mathrm{q}},c_{\mathrm{p}},c_{\mathrm{j}}) \leftarrow \mathsf{RS}(q_i,\mathbf{q},\mathbf{p},\mathbf{j},c_{\mathrm{q}},c_{\mathrm{p}},c_{\mathrm{j}})$
$\mathbf{r} \leftarrow (r_1,\ldots,r_n)$; $lk \leftarrow\!\!{}^{\$}\, \mathcal{L}(\mathbf{M},\mathbf{q})$; $S \leftarrow \bigcup_{q\in\mathbf{q}} \mathbf{M}[q]$
While $S \neq S'$ do $S \leftarrow S'$; $(\mathsf{SET},S',\mathbf{I}) \leftarrow \mathsf{IJ}(\mathbf{j},S',\mathbf{I})$
Define M : # of vals in $\mathbf{M}$ and $\mathbf{M}_1$
Define ℓ : max. length val in $\mathbf{M},\mathbf{M}_1$
Return $(\mathbf{r},lk,\mathsf{QP}(\mathbf{p}),c_{\mathrm{p}},\mathsf{HF}(\mathbf{p},S,\mathsf{SET}),|\mathsf{SET}'|,\mathbf{I},M,\ell,\mathsf{QP}(\mathbf{j}),c_{\mathrm{j}})$

Subroutine $\mathsf{RS}(q,\mathbf{q},\mathbf{p},\mathbf{j},c_{\mathrm{q}},c_{\mathrm{p}},c_{\mathrm{j}})$

If $q=(\mathtt{r},id)$ then $\mathbf{q} \xleftarrow{\cup} (\mathtt{r},id)$; $r \leftarrow (\mathtt{m},c_{\mathrm{q}})$; $c_{\mathrm{q}} \leftarrow c_{\mathrm{q}}+1$
Else if $q=(\mathtt{s},at,x,(\mathtt{r},id))$ then $\mathbf{q} \xleftarrow{\cup} (\mathtt{s},at,x)$; $r \leftarrow (\mathtt{m},c_{\mathrm{q}})$; $c_{\mathrm{q}} \leftarrow c_{\mathrm{q}}+1$
Else if $q=(\mathtt{s},at,x,q_1)$ then
  $(r_1,\mathbf{q},\mathbf{p},\mathbf{j},c_{\mathrm{q}},c_{\mathrm{p}},c_{\mathrm{j}}) \leftarrow \mathsf{RS}(q_1,\mathbf{q},\mathbf{p},\mathbf{j},c_{\mathrm{q}},c_{\mathrm{p}},c_{\mathrm{j}})$; $\mathbf{p} \xleftarrow{\cup} (\mathtt{s},at,x)$; $r \leftarrow (\mathtt{s},c_{\mathrm{p}},r_1)$; $c_{\mathrm{p}} \leftarrow c_{\mathrm{p}}+1$
Else if $q=(\mathtt{pp},\mathbf{t}_1,\mathbf{t}_2,q_1,q_2)$ then
  For $i=1,2$ do
    If $q_i=(\mathtt{r},id)$ then
      $\mathbf{q} \xleftarrow{\cup} (\mathtt{pp},\mathbf{t}_1,\mathbf{t}_2,i)$; $r_i \leftarrow (\mathtt{m},c_{\mathrm{q}})$; $c_{\mathrm{q}} \leftarrow c_{\mathrm{q}}+1$
    Else
      $(r'_i,\mathbf{q},\mathbf{p},\mathbf{j},c_{\mathrm{q}},c_{\mathrm{p}},c_{\mathrm{j}}) \leftarrow \mathsf{RS}(q_i,\mathbf{q},\mathbf{p},\mathbf{j},c_{\mathrm{q}},c_{\mathrm{p}},c_{\mathrm{j}})$
      $\mathbf{p} \xleftarrow{\cup} (\mathtt{pp},\mathbf{t}_1,\mathbf{t}_2,i)$; $r_i \leftarrow (\mathtt{p},c_{\mathrm{p}},r'_i)$; $c_{\mathrm{p}} \leftarrow c_{\mathrm{p}}+1$
  $r \leftarrow (\mathtt{pp},r_1,r_2)$
Else if $q=(\mathtt{fp},\mathbf{t}_1,\mathbf{t}_2,(\mathtt{r},id_1),(\mathtt{r},id_2))$ then $\mathbf{q} \xleftarrow{\cup} (\mathtt{fp},\mathbf{t}_1,\mathbf{t}_2)$; $r \leftarrow (\mathtt{m},c_{\mathrm{q}})$; $c_{\mathrm{q}} \leftarrow c_{\mathrm{q}}+1$
Else if $q=(\mathtt{fp},\mathbf{t}_1,\mathbf{t}_2,q_1,(\mathtt{r},id))$ or $q=(\mathtt{fp},\mathbf{t}_1,\mathbf{t}_2,(\mathtt{r},id),q_1)$ then
  $(r_1,\mathbf{q},\mathbf{p},\mathbf{j},c_{\mathrm{q}},c_{\mathrm{p}},c_{\mathrm{j}}) \leftarrow \mathsf{RS}(q_1,\mathbf{q},\mathbf{p},\mathbf{j},c_{\mathrm{q}},c_{\mathrm{p}},c_{\mathrm{j}})$
  If $q=(\mathtt{fp},\mathbf{t}_1,\mathbf{t}_2,q_1,(\mathtt{r},id))$ then $i=1$ else $i=2$
  $\mathbf{j} \xleftarrow{\cup} (\mathbf{t}_1,\mathbf{t}_2,i)$; $r \leftarrow (\mathtt{fp},c_{\mathrm{j}},i,r_1)$; $c_{\mathrm{j}} \leftarrow c_{\mathrm{j}}+1$
Else if $q=(\mathtt{fp},\mathbf{t}_1,\mathbf{t}_2,q_1,q_2)$ then
  For $i=1,2$ do $(r_i,\mathbf{q},\mathbf{p},\mathbf{j},c_{\mathrm{q}},c_{\mathrm{p}},c_{\mathrm{j}}) \leftarrow \mathsf{RS}(q_i,\mathbf{q},\mathbf{p},\mathbf{j},c_{\mathrm{q}},c_{\mathrm{p}},c_{\mathrm{j}})$
  $\mathbf{p} \xleftarrow{\cup} (\mathtt{fp},\mathbf{t}_1,\mathbf{t}_2)$; $r \leftarrow (\mathtt{iij},c_{\mathrm{p}},r_1,r_2)$; $c_{\mathrm{p}} \leftarrow c_{\mathrm{p}}+1$
Return $(r,\mathbf{q},\mathbf{p},\mathbf{j},c_{\mathrm{q}},c_{\mathrm{p}},c_{\mathrm{j}})$

Figure 2.17. Leakage algorithm used in Theorem 17, the proof of security for hybrid StI scheme HybStI. The subroutines $\mathsf{HF},\mathsf{IJ},\mathsf{QP}$ are given in Fig. 2.13.

Alg $\mathsf{EvalBW}(q)$

If $q = (\mathtt{r}, id)$ then $\mathbf{B}[(id)] \leftarrow \mathcal{N}(id)$
Else if $q = (\mathtt{s}, at, x, q_1)$ then
  $\mathbf{B} \leftarrow \mathsf{EvalBW}(q_1)$; $id \leftarrow \mathsf{getID}(at, \mathsf{scma})$
  $\mathbf{B}[(id)] \leftarrow \mathbf{B}[(id)] \cdot \mathcal{H}_{at}(x)$
Else if $q = (op, \mathbf{t}_1, \mathbf{t}_2, q_1, q_2)$ then
  $\mathbf{B} \leftarrow \mathsf{EvalBW}(q_1) \cup \mathsf{EvalBW}(q_2)$
  For $i = 1, 2$
    Define $\mathbf{i_i}$: $\mathsf{getID}(\mathbf{t}_i, \mathsf{scma}) \in \mathbf{i_i} \in \mathbf{B}.\mathsf{Lbls}$
  If $op = \mathtt{fp}$ then
    $N \leftarrow \frac{\mathcal{F}(\mathbf{t}_1, \mathbf{t}_2) \cdot \mathbf{B}[\mathbf{i_1}] \cdot \mathbf{B}[\mathbf{i_2}]}{\mathcal{N}(\mathsf{getID}(\mathbf{t}_1, \mathsf{scma})) \cdot \mathcal{N}(\mathsf{getID}(\mathbf{t}_2, \mathsf{scma}))}$
    $\mathbf{B}[\mathbf{i_1} \| \mathbf{i_2}] \leftarrow \mathbf{2} \cdot \mathbf{N}$
    $\mathbf{B}[\mathbf{i_1}] \leftarrow \perp$; $\mathbf{B}[\mathbf{i_2}] \leftarrow \perp$
  Else if $op = \mathtt{pp}$ then
    For $i = 1, 2$ do
      $\mathbf{B}[\mathbf{i_i}] \leftarrow \mathcal{P}_{\mathbf{i}}(\mathbf{t_1}, \mathbf{t_2}) \cdot \frac{\mathbf{B}[\mathbf{i_i}]}{\mathcal{N}(\mathsf{getID}(\mathbf{t_i}, \mathsf{scma}))}$
Return $\mathbf{B}$

Schema scma

Return $\mathsf{Schema}(\mathsf{DB})$

Table size $\mathcal{N}(id)$

Return $|\mathsf{DB}[id].\mathsf{T}|$

Freq. histogram $\mathcal{H}_{at}(x)$

$\mathsf{R} \leftarrow \sigma_{(at,x)}(\mathsf{DB}[\mathsf{getID}(at, \mathsf{scma})])$
Return $\frac{|\mathsf{R.T}|}{\mathcal{N}(id)}$

FP join size $\mathcal{F}(\mathbf{t}_1, \mathbf{t}_2)$

For $i = 1, 2$ do $id_i \leftarrow \mathsf{getID}(\mathbf{t}_i, \mathsf{scma})$
$\mathsf{R} \leftarrow \mathsf{DB}[id_1] \bowtie_{\mathbf{t}_1, \mathbf{t}_2} \mathsf{DB}[id_2]$
Return $|\mathsf{R.T}|$

PP join sizes $\mathcal{P}_j(\mathbf{t}_1, \mathbf{t}_2)$

For $i = 1, 2$ do $id_i \leftarrow \mathsf{getID}(\mathbf{t}_j, \mathsf{scma})$
$\mathsf{R} \leftarrow \mathsf{DB}[id_1] \bowtie_{\mathbf{t}_1, \mathbf{t}_2} \mathsf{DB}[id_2]$
Return $|\{\mathbf{r}[\mathsf{uk}(id_j)] : \mathbf{r} \in \mathsf{R.T}\}|$

Figure 2.18. EvalBW algorithm (left) defined in terms of precomputed statistics (right) stored on the client. Our heuristic assumes that q incurs bandwidth $\sum_{\mathbf{i} \in \mathbf{B}.\mathsf{Lbls}} \mathbf{B}[\mathbf{i}]$ where $\mathbf{B} = \mathsf{EvalBW}(q)$.

2. Maximize number of PP joins

3. If multiple choices remain, minimize bandwidth.

We argue that our setup is realistic because (1) we expect the J joins made in a query to be modest enough for the client to evaluate all 2^J HybDT queries, (2) bandwidth measurement can be reduced to the number of rows from ED sent as they are padded to the same length, and (3) is it common for SQL applications to limit bandwidth to prevent the client from maxing out its memory.

To complete this setup, we need a way for the client to estimate the bandwidth of a query with only partial information about **DB** computed during setup. These precomputed statistics are listed on the right of Fig. 2.18 and the bandwidth estimation algorithm is EvalBW. Intuitively, EvalBW will populate a dictionary $\mathbf{B}$ with entries $\mathbf{B}[\mathbf{i}]$ representing the bandwidth for the ciphertext set containing rows from all $\mathsf{DB}[id]$ where $id \in \mathbf{i}$. We estimate that a query $q \in \mathsf{HybDT.QS}$ incurs bandwidth $\sum_{\mathbf{i} \in \mathbf{B}.\mathsf{Lbls}} \mathbf{B}[\mathbf{i}]$ where $\mathbf{B} = \mathsf{EvalBW}(q)$. We will next explore the tradeoffs involved in storing these statistics.

Memory tradeoffs.

Notice that the client storage required for the precomputed statistics (as given in Fig. 2.18) increases with number of joins (i.e. $|\alpha|$) and size of histograms (i.e. $|\mathsf{rng}(at, \mathsf{DB})|$ for each at). In practice, data may be too complex or client devices may be too memory strapped (e.g. mobile devices) to store this in full. We describe two tradeoffs application designers can explore to better fit their system requirements.

When it is unfeasible to store full frequency histograms for some at, the client can partition $\mathsf{rng}(at, \mathsf{DB})$ into ranges and store this bucketed frequency histogram. EvalBW will approximate $\mathcal{H}_{at}(x)$ by assuming that values within a bucket are uniformly distributed. This approach is used in SQL server and the literature recommends 200 equiDepth (as opposed to equiWidth) buckets [127, 43]. In the extreme case, the client uses a single bucket and needs only store $|\mathsf{rng}(at, \mathsf{DB})|$ and uses $\mathcal{H}_{at}(x) \approx \frac{1}{|\mathsf{rng}(at, \mathsf{DB})|}$. Note that this only works when the elements of $\mathsf{rng}(at, \mathsf{DB})$ can be closely approximated and ordered. For example, this may not work with a "name" column because the names in $\mathsf{rng}(at, \mathsf{DB})$ are not dense in any easily enumerated set. In general, bucketing sacrifices the accuracy of EvalBW to reduce client memory. We study this tradeoff more in Section 2.6.

Above, we assumed the client would pre-compute and store the join sizes. When this is infeasible due to memory constraints, the client can alternatively compute join sizes using table sizes and the $\mathcal{H}_{at}(x)$ during EvalBW whenever $\mathsf{rng}(at)$ is enumerable. Notice that we can express each co-occurrence frequency as a function of the relevant occurrence frequencies. With a single attribute join, let $X = \mathsf{rng}(at_1, \mathsf{DB}) \cap \mathsf{rng}(at_2, \mathsf{DB})$, $N_i = \mathcal{N}(\mathsf{getID}(at_i, \mathsf{scma}))$ then

$$\mathcal{F}(at_1, at_2) = N_1 \cdot N_2 \cdot \sum_{x \in X} \mathcal{H}_{at_1}(x) \cdot \mathcal{H}_{at_2}(x) \quad \text{and} \quad \mathcal{P}_j(at_1, at_2) = N_j \cdot \sum_{x \in X} \mathcal{H}_{at_j}(x).$$

We can extend this to a cluster join $(\mathbf{t}_1, \mathbf{t}_2)$ where $\mathbf{t}_j = (at_1^j, \ldots, at_n^j)$. We substitute the above histogram values for $\mathcal{H}_{\mathbf{t}_j}(x_1, \ldots, x_n)$ and take the sum over all $(x_1, \ldots, x_n)$ where $x_i \in \mathsf{rng}(at_i^1, \mathsf{DB}) \cap \mathsf{rng}(at_i^2, \mathsf{DB})$. These frequencies are approximated by assuming that columns are independently

distributed: $\mathcal{H}_{\mathbf{t}_i}(x_1,\ldots,x_n) \approx \prod_{i\in[n]} \mathcal{H}_{at_i^j}(x_i)$. Note also that accuracy issues are compounded if frequency histograms are themselves estimated using bucketing. In general, approximating join sizes trades efficiency (of EvalBW) and accuracy (for cluster joins) to reduce memory.

2.6 Simulations on Real-World Datasets

To get some indication of how our schemes would fare in practice we simulate the storage and bandwidth they would incur in a real-world context. We show that in practice, PP indexing is likely to be more storage efficient than FP. We also confirm three claims made in this work: (1) PP indexing has equal or better bandwidth than FP on non-recursive joins (i.e. JnDT queries), (2) On recursive selects and joins (i.e. SjDT queries), the analogous choice is data and query dependent, and (3) our heuristic is accurate in finding optimal hybrid query execution plans.

We note that our goal here is not to make broad statements about all SQL data nor to perform a full system evaluation. We see our simulations more as a sanity check which might motivate large-scale implementations of our schemes. Additionally, we are not aware of any benchmarks with just join and select queries so we generate our own as described below.

Simulation data.

Our simulations uses data from the Chicago Open Data Portal and the MySQL Sakila benchmark. The Data Portal stores each Chicago relation separately and intends each relation to be useful on its own – independent from the other relations. The Sakila database also has 15 relations, with a total of 46,238 rows and 88 attributes. Unlike the Chicago database, the Sakila relations have a clear logical structure in the schema such that each relation has a role defined relative to the other relations. Details about the Sakila schema can be found on the MySQL website. By including one example database without a structured schema and one with, we hope to model two different use cases – one where the DBA treats each relation as existing independently and one where the DBA carefully pre-plans the entire organization.

To give an idea of the data distribution in our data sets, we give some summary statistics

Table 2.1. Summary statistics for the Chicago (left) and Sakila (right) data used in our simulations.

Chicago R name	|R.Ats|	|R.T|	*at* densities		
			Min	Ave	Max
Bike_Racks	12	5164	4e-4	0.53	1.0
Census_Data	9	78	0.72	0.91	1.0
Crimes_2019	30	1.7e5	6e-6	0.17	1.0
Employee_Debt	7	1.4e4	3e-3	0.10	0.46
Fire_Stations	7	92	0.01	0.65	1.0
Grafitti	5	67	0.09	0.68	1.0
Housing	14	915	0.03	0.32	0.56
IUCR_Codes	4	404	5e-3	0.51	1.0
Land_Inventory	19	2.0e5	3e-4	0.41	1.0
Libraries	9	81	0.01	0.63	1.0
Lobbyists	7	1537	0.03	0.22	0.66
Police_Stations	15	23	0.04	0.87	1.0
Reloc_Vehicles	20	2672	2e-3	0.51	1.0
Street_Names	7	2582	2e-3	0.31	1.0
Towed_Vehicles	10	3339	1e-3	0.19	0.99

Sakila R name	|R.Ats|	|R.T|	*at* densities		
			Min	Ave	Max
Address	8	603	2e-3	0.93	1.0
Actor	4	208	0.02	0.56	1.0
Category	3	16	0.06	0.69	1.0
City	4	600	2e-3	0.55	1.0
Country	3	109	0.01	0.67	1.0
Customer	10	599	2e-3	0.5	1.0
Film	14	1002	1e-3	0.37	1.0
Film_Actor	3	5462	1e-4	0.07	0.18
Film_Cat	3	1000	1e-3	0.34	1.0
Inventory	4	5481	2e-4	0.30	1.0
Language	3	6	0.17	0.72	1.0
Payment	7	1.6e4	1e-4	0.44	1.0
Staff	11	2	0.5	0.86	1.0
Store	4	2	0.5	0.88	1.0
Rental	7	1.6e4	1e-4	0.47	1.0

about each in Table 2.1. We report each relation's name, number of attributes, number of rows, and minimum, mean, and maximum attribute densities. The density of an attribute is the average occurrence frequency of the values in that column. In other words, for relation $\mathsf{DB}[id]$ *at*'s attribute density is $\frac{|\mathsf{rng}(at)|}{|\mathsf{DB}[id].\mathsf{T}|}$.

Simulation setup.

Our simulation dataset uses all relations from the MySQL Sakila benchmark [1] and the following fifteen frequently accessed relations from Chicago's Open Data Portal:

Bike_Racks, Census_Data, Crimes_2019, Employee_Debt, Fire_Stations, Grafitti
, Housing, IUCR_Codes, Land_Inventory, Libraries, Lobbyists, Police_Stations
, Reloc_Vehicles, Street_Names, Towed_Vehicles.

In total, our setup involved 30 relations, 175 attributes and 219,992 rows. We provide a full, annotated source code for our simulations in [119].

We include in α all single-attribute joins that return at least one row. This helps to filter out meaningless join queries (e.g. joining on "language" and "actor"). We consider joins within the Sakila relations and joins within the Chicago relations, but we do not attempt joins between the two independent sources. We generate recursive queries with J joins and S selections by selecting uniformly at random J distinct joins from α as well as S attributes and elements of their domains, discarding queries that return no rows. When $J \geq 2$ we only use input tables with less than 1000 rows to avoid very large output relations.

Server storage.

With the above setup we can get an idea of how much server-side storage would be required by each of our indexing schemes. Recall that our schemes make use of a RR multimap primitive and/or a hashset filtering primitive. Therefore, in Fig. 2.2 we report the number of multimap [2] labels and values as well as the values in hashset HS for each of our StI schemes.

[1] We excluded the film_text relation since it is a subset of the film relation

[2] Note that in the case of FpSj, HybStI, this includes the multimap for internal joins.

Table 2.2. Simulated server storage for each data set using each of our schemes in terms of multimap (MM) labels/ values and hashset (HS) values broken down by the query type being indexed (i.e. relation retrievals, non-recursive/ recursive joins, or selections).

Query Type	Indexed Data	Chicago data set					Sakila data set				
		JnDT		SjDT		HybDT	JnDT		SjDT		HybDT
		FpJn	PpJn	FpSj	PpSj	HybStl	FpJn	PpJn	FpSj	PpSj	HybStl
Non-recursive join	MM lbls	1249	2498	1249	2498	3747	631	1262	631	1262	1893
	MM vals	1.495e10	2.796e7	1.495e10	2.796e7	1.498e10	5.103e8	2.201e6	5.103e8	2.201e6	5.125e8
Recursive join	MM lbls	–	–	2.796e7	–	2.796e7	–	–	2.202e6	–	2.202e6
	MM vals	–	–	1.496e10	–	1.496e10	–	–	5.107e8	–	5.107e8
	HS vals	–	–	7.477e9	2.796e7	7.505e9	–	–	2.552e8	2.201e6	2.574e8
Relation retrieval	MM lbls	–	–	15	15	15	–	–	15	15	15
	MM vals	–	–	4.010e5	4.010e5	4.010e5	–	–	4.409e4	4.409e4	4.409e4
Select	MM lbls	–	–	1.082e6	1.082e6	1.082e6	–	–	1.190e5	1.190e5	1.190e5
	MM vals	–	–	5.749e6	5.749e6	5.749e6	–	–	2.945e5	2.945e5	2.945e5
	HS vals	–	–	5.749e6	5.749e6	5.749e6	–	–	2.945e5	2.945e5	2.945e5
Total	MM lbls	1249	2498	2.905e7	1.085e6	2.905e7	631	1262	2.321e6	1.203e5	2.322e6
	MM vals	1.495e10	2.796e7	2.991e10	3.412e7	2.994e10	5.103e8	2.201e6	1.021e9	2.540e6	1.023e9
	HS vals	–	–	7.483e9	3.371e7	7.511e9	–	–	2.555e8	2.496e6	2.577e8

We present our simulation results for the two datasets separately since the Chicago data set contains many more rows and would dominate the Sakila statistics. Additionally, we also show a breakdown of these statistics in terms of the queries they index to better understand the cost of each type of query support.

A number of observations can be made from this data. In our simulation we see that even though there are more selections to index (as evidenced by the number of labels), the multimap size (i.e. number of values) is dominated by join indexes. We expect this cost to be lower in a real system because a judicious database administrator can reduce the set of supported joins (α) to a smaller number than we did. Our simulation also brings forth another advantage of PP join indexing – it is more storage efficient by several orders of magnitude. This is because each row token is stored at most once per join (the same thing which causes PP to have better bandwidth) and, in the case of SjDT, there is no need for the "internal join" indexing which essentially doubles the multimap's labels and values. Finally, for the above reason, the storage overhead of hybrid indexing over FP indexing is very small so systems which currently use indexing schemes like FP (e.g. OPX or SPX) can upgrade its security at low cost.

Join categories.

We partition joins into three classes which behave quite differently: *one-one*, *one-many* and *many-many*. We say that a join $\mathsf{R} \leftarrow \mathsf{R}_1 \bowtie_{at_1,at_2} \mathsf{R}_2$ is one-one if each row in $\mathsf{R}_1, \mathsf{R}_2$ occurs at most once in R. It is one-many if the above is true for one relation but not for the other. It is

Table 2.3. Breakdown of all possible non-recursive join queries which returns at least one row by join types. For each type, we simulated the number of rows that would be sent using FP and PP indexing, and report the minimum, average and maximum overhead incurred.

Join category	# joins	Ratio of FpJnto PpJn BW		
		Min	Ave	Max
One-one	237	1.0	1.0	1.0
One-many	711	1.0	1.8	2.0
Many-many	932	1.5	465	8000

many-many if there exists rows in both $\mathsf{R}_1, \mathsf{R}_2$ which occur more than once in R. We record the breakdown of these classes in our datasets in Fig. 2.3.

StI for JnDT**.**

In Section 2.4 we showed that PP indexing has superior bandwidth on non-recursive join queries. We demonstrate that these savings by computing all 1880 possible joins in α and report our findings in Fig. 2.3. As one would expect, PP indexing always performs equal or better to FP – they perform equally for one-one joins but there are moderate and significant savings for one-many and many-many joins respectively.

StI for SjDT**.**

In Section 2.4.2 we noted that neither PP nor FP joins are strictly superior when it comes to recursive SjDT queries. We demonstrate this using our datasets. For each combination of 1 to 3 joins and 0 to 2 selects, we randomly sampled 25 queries and report the results in Fig. 2.4. As can be seen, neither scheme can reliably achieve the optimal bandwidth. While FpSj performed better on average, its maximum overhead exceeds that of PpSj in about half the cases.

Hybrid StI.

In Section 2.5 we provided a heuristic for client-side leakage-aware query planning. We demonstrate its efficacy when frequency histograms are estimated via three bucketing options: $B = |\mathsf{rng}(at, \mathsf{DB})|$ (full histograms), $B = 200$ and $B = 1$. We use the same 225 queries as the SjDT simulations and set the bandwidth limit L for each $q \in \mathsf{SjDT}$ to be the mean incurred by all

2^J possible HybDT queries to ensure that the optimization is non-trivial. Additionally, join sizes $\mathcal{F}, \mathcal{P}_1, \mathcal{P}_2$ are estimated using the histogram. Therefore, our simulation is conservative and we expect our heuristic to perform better in applications with a fixed L and precomputed join sizes.

In Fig. 2.5 we show how our heuristic performed for each query type and histogram estimation technique. When the optimal join annotation is not returned we note which "level" the heuristic failed at, where the levels are defined in relation to our definition of "optimality" given in Section 2.5. In particular, an R1 failure means the returned q' exceeds bandwidth limit L when StE.Eval is run, an R2 failure means q' used more FP joins than was necessary to reduce bandwidth below L and an R3 failure means q' was not the smallest bandwidth option which uses the minimal number of FP joins while meeting L.

Unsurprisingly, there is a direct tradeoff between client memory and the heuristic's accuracy: across all 225 queries, the heuristic returned the optimal q' on 198 with full histograms but only 143 and 76 when $B = 200$ and $B = 1$ respectively. More interestingly, our heuristic seems to improve when the search space increases: when there is one join the heuristic performed slightly better averaged across all three B values than guessing (58.7% vs 50%) but when there are three it performs significantly better (56.4% vs 12.5%). This demonstrates that our heuristic works when it is most needful since we expect the bandwidth overhead from an incorrect choice to increase with query complexity.

2.7 Conclusion

Our work introduces partially precomputed join indexing and incorporates it into a hybrid StE scheme. While we did not explore it in this work, we believe that our schemes can be extended to support dynamic queries and adaptive security via multimap primitives of the same kind. We believe the former can be achieved in a similar way to KM's extension of SPX to SPX$^+$. To achieve the latter, our schemes can be reframed in JN's model for adaptive compromise [85]. Future work can also extend our query support, possibly by incorporating cryptographic

Table 2.4. On randomly generated queries involving the indicated number of joins ($\bowtie$) and selects (σ), we report the minimum, average and maximum ratios of rows sent using each indexing technique compared to the theoretical minimum possible.

Query type	Ratio of BW to ideal					
	FpSj			PpSj		
	Min	Ave	Max	Min	Ave	Max
1 $\bowtie$, 0 σ	1.0	9.6	37	1.0	1.0	1.0
1 $\bowtie$, 1 σ	1.0	1.6	4.0	1.0	60	302
1 $\bowtie$, 2 σ	1.0	1.3	2.0	1.0	90	500
2 $\bowtie$, 0 σ	1.0	3.3	57	1.3	13	54
2 $\bowtie$, 1 σ	1.0	15	201	1.0	41	201
2 $\bowtie$, 2 σ	1.0	14	121	1.0	93	535
3 $\bowtie$, 0 σ	1.0	7.2	48	2.4	9.1	17
3 $\bowtie$, 1 σ	1.0	6.5	63	2.6	23	60
3 $\bowtie$, 2 σ	1.0	5.0	61	2.3	30	84

Table 2.5. On randomly generated queries involving the indicated number of joins ($\bowtie$) and selects (σ), we report the accuracy of our heuristic under different client storage. When a suboptimal query execution plan is returned, we report the point at which our heuristic fails (with R3 being the closest to success).

Query type	Bucketed $B = 1$				Bucketed $B = 200$				Full histograms			
	Correct	Wrong			Correct	Wrong			Correct	Wrong		
		R1	R2	R3		R1	R2	R3		R1	R2	R3
1 $\bowtie$, 0 σ	14	11	0	0	25	0	0	0	25	0	0	0
1 $\bowtie$, 1 σ	6	17	0	6	12	0	12	1	16	0	8	1
1 $\bowtie$, 2 σ	5	0	0	20	14	0	1	10	15	0	0	10
2 $\bowtie$, 0 σ	5	0	0	20	21	3	0	1	25	0	0	0
2 $\bowtie$, 1 σ	15	0	1	9	18	6	1	0	24	0	1	0
2 $\bowtie$, 2 σ	17	8	0	0	20	0	1	4	23	0	0	2
3 $\bowtie$, 0 σ	5	20	0	0	8	10	7	0	21	2	2	0
3 $\bowtie$, 1 σ	2	23	0	0	19	1	5	0	25	0	0	0
3 $\bowtie$, 2 σ	7	18	0	0	16	4	5	0	24	0	1	0

techniques for range queries or aggregations [63, 78]. Higher query support would also enable more rigorous testing using real-world applications and query benchmarks. Stronger security can be achieved using lower-leakage indexing primitives [88, 114, 91].

We also introduce leakage-aware query planning which we believe to be of independent interest as it incorporates structured indexing into DBMS architecture, which may help StE become a part of commercial DBMSes. Future work could improve our heuristic's efficiency and accuracy, or develop analogous hybrid schemes for other query classes.

2.8 Acknowledgements

We thank Mihir Bellare and Francesca Falzon for discussions and insights.

This chapter, in full, is a reprint of the material as it appears in International Conference on Applied Cryptography and Network Security – ACNS 2021. Cash, David; Ng, Ruth; Rivkin, Adam, Springer Lecture Notes in Computer Science volume 12727, 2021. The dissertation author was the primary investigator and author of this paper.

Chapter 3

Composition of Structured Encryption and its Relation to Key-Dependent Security

3.1 Introduction

Structured encryption (StE) [52] allows one to encrypt a data structure and then delegate the ability to run queries via query-specific tokens. While many techniques can be fit into the definitional framework of StE, much research has been on simple, efficient constructions from basic symmetric encryption with few rounds of interaction. This efficiency is enabled by allowing for some controlled leakage to the party holding the encrypted data structure, which may include sizes, access patterns, and other information. Formally, one can use simulation-style definitions, where the simulator is given the output of a so-called *leakage algorithm*.

In their work introducing the StE framework, Chase and Kamara gave a vision building more advanced StE via *composition*. Recent work has leveraged composition to support large and complex subsets of SQL queries on several tables [89, 92]. As a simpler example, consider the case of an encrypted file system which allows document retrieval via keywords. This can be implemented using two StE schemes $\mathsf{StE}_1, \mathsf{StE}_2$, where StE_1 will map keywords to lists of document identifiers, and StE_2 will map these identifiers to document payloads. Using composition, one can assemble them into a larger StE which takes keywords and returns documents

(in Section 3.3 we formalize this as *double-level indexing* and treat it in detail). To prove the security of this aggregate construction, one builds a simulator from the simulators for $\mathsf{StE}_1, \mathsf{StE}_2$ with leakage that depends on the leakages of StE_1 and StE_2. The utility of this generic composition result, as Chase and Kamara point out, is that the component StEs can be swapped out to fit the application or as new constructions are designed. For example, recent *volume-hiding* constructions [90, 114] might be used without requiring new proofs.

This paper.

We consider composition of StE schemes, and in particular of StE for dictionaries and multimaps. We begin by observing that standard semantic security for StE does not enable some straightforward reductions that may appear to work at first glance. The issue arises when the tokens of one StE are stored in another (or in another instance of itself), like in the double-level indexing problem above. The technical problem is that the reduction must run a pair of simulators that need to work together. In Section 3.4 we provide a minimal example of a proof exhibiting this problem, and identify some steps in prior work that exhibit this gap.

We then address the double-level indexing problem in two ways. First, we give an extra condition which we call *content obliviousness* for a leakage profile, and show that the prior proof approach can be recovered for such leakage functions. Roughly, a content oblivious leakage algorithm will ensure that inputs with the same "shape" will have the same leakage profile. This resolves the coordination problem because our larger simulator can now select and input values into the simulators of its primitives to ensure they return consistent query outputs. We also show that this condition is easy to satisfy, and that most StE constructions have content oblivious leakage.

Our second approach is to give a *monolithic* solution to the double-level indexing problem. Our idea is to avoid the division of data into multiple StEs when possible, and instead pre-process data into a single, monolithic data structure and use a single StE. To compute a query, the evaluator will actually query the monolithic StE multiple times, feeding outputs back in as

inputs. In addition to possibly being more efficient than managing multiple StEs, the monolithic construction will be shown to have strictly less leakage since it leaks only the aggregate total size of the data, rather than the sizes of two component data structures, a distinction that can easily be meaningful in practice.

In analyzing the monolithic construction, another proof challenge comes up: existing definitions of semantic security for StE do not allow for the storage of data (like tokens) that depend on the secret key used for encryption. Such issues have arisen with symmetric- and public-key encryption, and have been well studied under various notions of key-dependent (KD) message security. We adapt this line of thinking to the StE for dictionaries/multimaps, with a new definitions of KD security, and then use our KD notion to analyze our monolithic solution. We show that many state-of-the-art dictionary/multimap StE primtives with pseudorandomly generated tokens achieve this notion of KD security with no additional assumptions, meaning that they can be securely adapted for use in the monolithic construction.

KD security of StE may be of interest beyond enabling constructions like ours. Systems may choose to manage access control by inserting keys into the data structure being encrypted, or may do so accidentally (say if StE is used to manage disk backups, and the key may have been swapped to disk).

Thus, in the final part of this paper we provide a set of foundational results. We observe that KD security behaves differently for response-revealing (RR) and response-hiding (RH) dictionary/ multimap StE, due to the revelation of key-dependent responses, and thus investigate both in detail.

In the case of RH, we show that "full" KD security is impossible without the significant storage and bandwidth overheads of volume-hiding primitives and ORAM. However, when we restrict the KD adversary to key-independent labels and fixed-length values (we formalize this as "fixed format" outputs in Section 3.6) this can be achieved with non-trivial efficiency. We demonstrate this with a general transform using KD-secure symmetric encryption as a primitive, and also that KD-security can come "for free" if KD-secure encryption is used in specific

dictionary/ multimap StE schemes from the literature when the underlying encryption is KD secure. In the case of RR, however, KD security is impossible for even the restricted class we identify above (thereby implying that "full" KD security is also impossible). In both the RH and RR cases, our results are more subtle than those of symmetric encryption because we need to handle edge-case schemes with extreme leakage profiles. For example, the impossibility of "full" KD-security in the RR case may be intuitively true (given that the adversary can see unencrypted values from the dictionary/ multimap) but given a trivial StE scheme which performs no encryption (accompanied by a leakage profile which leaks everything) KD security is indeed feasible (but meaningless).

Related work.

StE was first introduced by CK [52] as a generalization of symmetric searchable encryption which was first introduced by SWP and formalized by CGKO [128, 56]. The StE framework can and has been used to capture many real-world use cases including encrypting SQL data [89, 92] and supporting rich keyword queries in document storage systems [48, 49, 132, 63].

Added functionality and security has been studied for specific forms of StE, including support for dynamic data structures [94, 93], volume hiding queries [90, 112, 111], models for adaptive compromise [85], costs of minimizing leakage [113, 91] and many more [67, 22, 50, 129, 40, 11, 41, 87, 7, 12, 58].

StE has been subject to so-called *leakage-abuse attacks* which can sometimes recover damaging information about queries and encrypted data [83, 110, 47, 117, 135]. The attacks work against proven-secure constructions by exploiting the permitted leakage, so they are independent of possible gaps in proofs due to composition. However reducing leakage in order to limit leakage abuse has been a common goal.

Key dependent message (KDM) security was first introduced in BRS [36]. The special case of circular security was subsequently studied, mainly for public-key encryption [46, 38, 6, 10, 42, 15, 69, 97, 68]. Historically, KD security has focused on applications such as encrypting

a key on a disk and other forms of circular security.

3.2 Preliminaries

We denote the empty string with ε and the empty tuple with $()$. Given positive integer n, let $[n] = \{1, \ldots, n\}$. Given a set S or tuple $\mathbf{t}$, we write $S \xleftarrow{\cup} x$ as a shorthand for $S \leftarrow S \cup \{x\}$ and $\mathbf{t} \xleftarrow{\cup} x$ for $\mathbf{t} \leftarrow \mathbf{t} \| (x)$. Given a string $s \in \{0,1\}^*$ and integer len we write $s \leftarrow \langle s \rangle_{\mathsf{len}}$ to pad s's length up to a multiple of len and write $s_1 \| \ldots \| s_n \leftarrow s$ to parse it into blocks of that length.

Pseudocode.

In pseudocode, we will assume that all integers, tuples, strings and sets and arrays are initialized to 0, $()$, ε and $\emptyset$ respectively. We also often present pseudocode for multiple algorithms at the same time, indicating differences in the code with boxes. In this case, unboxed code belongs to all algorithms and code in either a $\boxed{\text{solid}}$ or $\boxed{\text{dashed}}$ box belongs only to the respective algorithm. (For an example of this, see the games for SE in Fig. 3.1).

Additionally, we will "Define X : *pred*" to set X (a function or constant) in such a way that the predicate *pred* is true. If there are undefined variables in *pred* we treat it as a random variable and expect that X is defined such that *pred* will always be true.

Games.

We use the code-based game-playing framework of BR [30]. Given oracle O and adversary $\mathcal{A}$, we write $x \leftarrow\!\!\$\, \mathcal{A}^{\mathsf{O}}(x_1, \ldots, x_m)$ to denote that $\mathcal{A}$, a possibly randomized algorithm, is run with inputs $x_1, \ldots, x_m$ and its output is x. It has black-box access to O and can make as many queries as it likes. Given game G we write $\Pr[\mathrm{G}(\mathcal{A})]$ to denote the probability that $\mathcal{A}$ plays G and the latter returns true. If G contains pseudocode saying "Require *pred*", it means that the game will evaluate the predicate *pred* at that point and if it returns false so will G (i.e. the adversary automatically loses).

Data Types, Structured Encryption.

The following definitions follow CK's formalism [52].

A data type DT defines domain set $\mathsf{DT.Dom}$, query set $\mathsf{DT.QS}$, and a deterministic specification function $\mathsf{DT.Spec} : \mathsf{DT.Dom} \times \mathsf{DT.QS} \to \{0,1\}^* \cup \{\perp\}$.

A structured encryption scheme for DT defines a non-empty key set $\mathsf{StE.KS}$ and the following algorithms:

- Randomized encryption algorithm $\mathsf{StE.Enc}$ which takes as input a data structure $DS \in \mathsf{DT.Dom}$ and a key $K \in \mathsf{StE.KS}$. It returns an encrypted data structure $ED \in \{0,1\}^*$.
- Possibly randomized token generation algorithm $\mathsf{StE.Tok}$ which takes as input a key and a query $q \in \mathsf{DT.QS}$, and it returns fixed length token $tk \in \{0,1\}^{\mathsf{StE.tl}}$.
- Deterministic secure evaluation algorithm $\mathsf{StE.Eval}$ which takes as input a token and an encrypted data structure, and returns a ciphertext $c \in \{0,1\}^*$ or $\perp$.
- Deterministic decryption algorithm $\mathsf{StE.Dec}$ which takes a key and a ciphertext, and returns a query output $s \in \{0,1\}^*$ or $\perp$.

The correctness condition is that $\Pr[\mathsf{StE.Dec}(K,c) = \mathsf{DT.Spec}(DS,q)] = 1$ where the probability is taken over all $K \in \mathsf{StE.KS}$, $DS \in \mathsf{DT.Dom}$ and $q \in \mathsf{DT.QS}$ and the random variables are defined via $ED \leftarrow\!\!\$\, \mathsf{StE.Enc}(DS)$, $tk \leftarrow\!\!\$\, \mathsf{StE.Tok}(K,q)$, and $c \leftarrow \mathsf{StE.Eval}(tk, ED)$.

StE schemes are usually classified into two *response types*: response revealing (RR) and response hiding (RH). In a RR scheme the server learns the query result from the evaluation algorithm. In other words, for all $DS \in \mathsf{DT.Dom}$ and $q \in \mathsf{DT.QS}$ we have that $\mathsf{DT.Spec}(DS,q) = c = \mathsf{StE.Dec}(K,c)$ (where the random variables are defined as in the correctness condition). Any scheme that is not RR is RH[1].

While we allow $\mathsf{StE.Eval}$ and $\mathsf{StE.Dec}$ to return $\perp$, this is to handle malformed input. Unless otherwise stated, we will leave implicit the handling of such in StE's pseudocode and

[1] In Section 3.5 we introduce a third response type: response flexible StE.

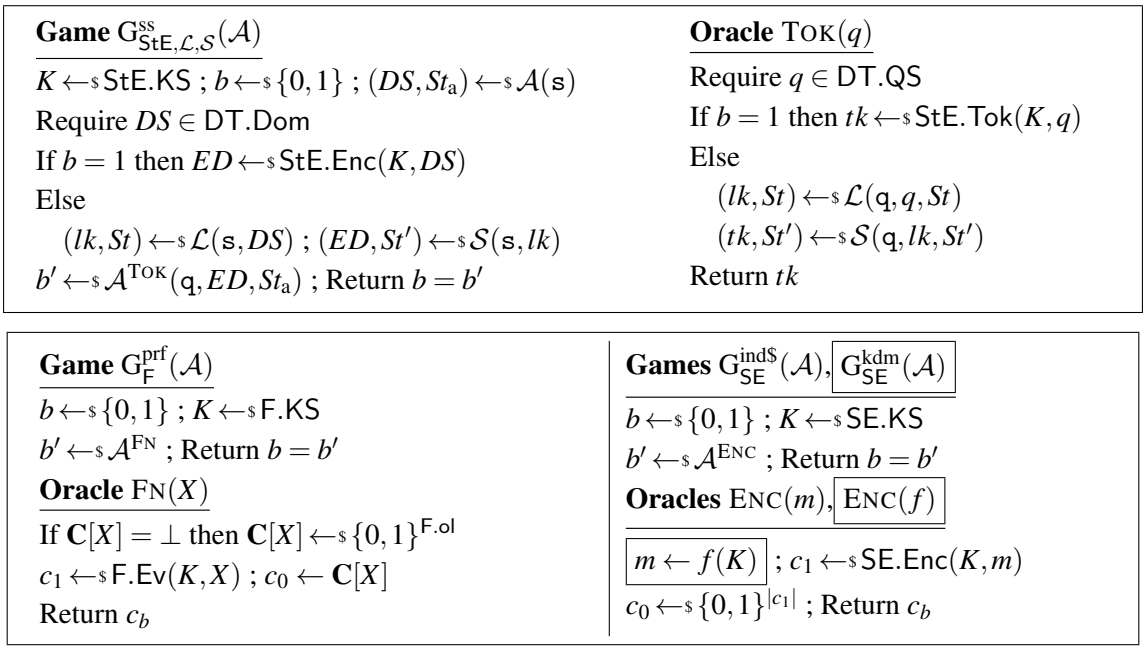

Figure 3.1. Games used in defining adaptive semantic security of StE (top), PRF security of function family F (bottom left) and IND$-security or KDM-security of symmetric encryption scheme SE (bottom right). Here, $\mathcal{A}$ is an adversary, StE is a structured encryption scheme for data type DT, $\mathcal{L}$ is a leakage algorithm and $\mathcal{S}$ is a simulator.

assume that adversaries do not make queries which will trigger this behavior.

Semantic security.

CK defines adaptive semantic security for StE using game $G^{ss}_{StE,\mathcal{L},\mathcal{S}}$ where $\mathcal{L}, \mathcal{S}$ are the leakage algorithm and simulator respectively. In $G^{ss}_{StE,\mathcal{L},\mathcal{S}}(\mathcal{A})$, all three algorithms (i.e. $\mathcal{A}, \mathcal{L}, \mathcal{S}$) have a setup phase and a query phase. We use the first argument to the algorithm – s or q – as a flag to indicate the phase to run in. The details of $G^{ss}_{StE,\mathcal{L},\mathcal{S}}$ are given in Fig. 3.1. The advantage of $\mathcal{A}$ is $\mathbf{Adv}^{ss}_{StE,\mathcal{L},\mathcal{S}}(\mathcal{A}) = 2\Pr[G^{ss}_{StE,\mathcal{L},\mathcal{S}}(\mathcal{A})] - 1$.

The above security definition applies to both RR and RH StE schemes. Notice that with a RR scheme, in order for $\mathcal{S}$'s tokens to be indistinguishable from those generated with StE.Tok the leakage must reveal the query responses (i.e. $DT.Spec(q, DS)$) to $\mathcal{S}$ for it to run $\mathcal{S}(q, q, St_a)$. Therefore, we will assume WLOG that if $(lk, St) \leftarrow\$ \mathcal{L}(q, q, St)$ then $lk = (DT.Spec(q, DS), lk')$ for some lk'.

Function families, PRF security.

A function family F defines a key set $\mathsf{F.KS}$ and output length $\mathsf{F.ol}$. It defines an evaluation algorithm $\mathsf{F.Ev} : \mathsf{F.KS} \times \{0,1\}^* \to \{0,1\}^{\mathsf{F.ol}}$. We define PRF security for function family F via the game $\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}$ depicted in Fig. 3.1. Given adversary $\mathcal{A}$, let $\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{A}) = 2\Pr[\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{A})] - 1$ be its PRF advantage.

Symmetric Encryption, IND$ and KDM security.

A symmetric encryption scheme SE defines key set $\mathsf{SE.KS}$, encryption algorithm $\mathsf{SE.Enc}$ and decryption algorithm $\mathsf{SE.Dec}$. and ciphertext length function $\mathsf{SE.cl}$. We require that if $C \leftarrow\!\!\$\, \mathsf{SE.Enc}(K,M)$ then $|C| = \mathsf{SE.cl}(|M|)$ and $\mathsf{SE.Dec}(K,C) = M$.

We define two notions of security for SE: IND$ and KDM [2] security depicted in games $\mathrm{G}^{\mathrm{ind\$}}_{\mathsf{SE}}$, $\mathrm{G}^{\mathrm{kdm}}_{\mathsf{SE}}$ in Fig. 3.1. In the latter game, we require that f provided to the ENC oracle is a "fixed-length" function, meaning that $|f(K_1)| = |f(K_2)|$ for all $K_1, K_2 \in \mathsf{SE.KS}$. Given adversary $\mathcal{A}$, let $\mathbf{Adv}^{\mathrm{ind\$}}_{\mathsf{SE}}(A) = 2[\Pr[\mathrm{G}^{\mathrm{ind\$}}_{\mathsf{SE}}(\mathcal{A})] - 1$ be its IND$ advantage and $\mathbf{Adv}^{\mathrm{kdm}}_{\mathsf{SE}}(A) = 2\Pr[\mathrm{G}^{\mathrm{ind\$}}_{\mathsf{SE}}(\mathcal{A})] - 1$ be its KDM advantage.

3.3 StE for Double-Level Indexing

As discussed in Section 3.1, we want to model a system with an index mapping indexing labels to payload labels where the latter are each associated with a payload value. We do so within CK's StE framework, which we reviewed in Section 3.2, using the new *double-level indexing* data type DLdt. We also introduce the *array data type* Adt which subsumes the "dictionary" and "multimap" data types of prior work and discuss how array encryption (StE for Adt) can be achieved using known techniques. These array encryption (AYE) schemes will be used as building blocks for our DLdt StE constructions.

[2] This KDM definition is slightly stronger than the definition given by BRS [36], because we require indistinguishablity from random strings, while their definition only requires indistinguishability from encryptions of $0^{|m|}$.

Array Data Type.

We define an array $\mathbf{A}$ as a mapping from labels $\ell \in \{0,1\}^*$ to values $\mathbf{A}[\ell] \in \{0,1\}^* \cup \{\perp\}$. We define $\mathbf{A}$'s "label set" to be the set of labels not mapping to $\perp$, which we write as $\mathbf{A}.\mathsf{Lbls} = \{\ell \in \{0,1\}^* : \mathbf{A}[\ell] \neq \perp\}$. We define $\mathbf{A}.\mathsf{Vals} = \{\mathbf{A}[\ell] : \ell \in \mathbf{A}.\mathsf{Lbls}\}$ as the analogous "value set". In pseudocode, all entries in an arrays are assumed to be initialized to $\perp$.

We now formalize arrays as a data type Adt. This data type defines a fixed block length bLen such that for all $\mathbf{A} \in \mathsf{Adt.Dom}$ and $\ell \in \mathbf{A}.\mathsf{Lbls}$ we have $|\mathbf{A}[\ell]| = n \cdot \mathsf{bLen}$ for some integer n. Then, we define $\mathsf{Adt.QS} = \{0,1\}^*$ and $\mathsf{Adt.Spec}(\mathbf{A},\ell) = \mathbf{A}[\ell]$. We will assume that all arrays in this work belong to $\mathsf{Adt.Dom}$ and that bLen is fixed.

We note that arrays can be used to store arbitrary data so long as it can be encoded to and parsed from bitstrings whose length is a multiple of bLen. In this work, we sometimes store sets as array values, leaving implicit the encoding, parsing and padding. We stress that this is distinct from interpreting array values as blocks of length bLen which is sometimes necessary in our schemes (i.e. $B_1\|\ldots\|B_n \leftarrow \langle \mathbf{A}[\ell] \rangle_{\mathsf{bLen}}$).

This definition serves as a generalization of dictionary and multimap data sets formalized in the literature (e.g. in [89]). The former is a mapping from labels to strings and depending on the formalism either assumes $\mathsf{bLen} = 1$ meaning all values are allowed or $\mathsf{bLen} = \max_{v \in \mathbf{A}.\mathsf{Vals}} |v|$ meaning the values are all padded to the maximum size. Multimaps are a mapping from labels to sets or tuples which we interpret as strings and interpret this string in terms of blocks.

Intuitively, when these arrays are encrypted bLen presents a tradeoff between storage overhead and security. Notice that a larger bLen may result in extra padding when translating data into array values, but will also result in less volume leakage (as discussed in Section 3.5, we assume that when a query is made, schemes leak the number of blocks returned). In practice, we expect bLen to be chosen in an application-specific manner.

We also use bLen to define what it means for two arrays to be "similar". Intuitively, this means that they have the same label set and that values under the same label have the same length. More formally, we say that two array $\mathbf{A}, \mathbf{A}'$ are *homomorphic* if $\mathbf{A}.\mathsf{Lbls} = \mathbf{A}'.\mathsf{Lbls}$ and

$|\mathbf{A}[\ell]| = |\mathbf{A}'[\ell]|$ for all $\ell \in \mathbf{A}.\mathsf{Lbls}$.

Array encryption.

We refer to StE for Adt as array encryption (AYE). Techniques in the literature handling Searchable Symmetric Encryption (SSE), dictionary encryption or multimap encryption can all be applied to achieve semantically secure AYE. We use AYE schemes as a primitive to build more complex StE.

None of our schemes will require specific AYE primitives, nor specific leakage profiles. However, it is useful to contextualize the security of our schemes using the leakage profile of state-of-the-art schemes from the literature. Intuitively, the setup leakage of this profile reveals the total number of blocks among all values in the array. The query leakage includes the query and access patterns. The former is the equality pattern of all the queries made thus far. In a RR scheme the latter is just the query response but in a RH scheme the latter is the number of blocks returned.

Now we detail these leakage algorithms and give example AYE schemes that achieve it derived from CJJ+'s $\prod_{\text{bas}}$ SSE scheme [48] and 2Lev from the Clusion library [101]. In Fig. 3.2, $\mathcal{L}^{\pi}_{\text{r}}$ (resp. $\mathcal{L}^{\pi}_{\text{h}}$) is the RR (resp. RH) leakage algorithm for $\mathsf{Aye}^{\pi}_{\text{r}}$ (resp. $\mathsf{Aye}^{\pi}_{\text{h}}$). The primitives used in $\mathsf{Aye}^{\pi}_{\text{r}}, \mathsf{Aye}^{\pi}_{\text{h}}$ are symmetric encryption scheme SE and function family F. We require that $\mathsf{SE.KS} = \{0,1\}^{\mathsf{F.ol}}$. Note that $\mathsf{Aye}^{\pi}_{\text{h}}.\mathsf{KS} = \mathsf{SE.KS} \times \mathsf{F.KS}$ and $\mathsf{Aye}^{\pi}_{\text{r}}.\mathsf{KS} = \mathsf{F.KS}$.

Double-level indexing.

In this data type, each domain element is a tuple of arrays we call the *payload* and *indexing* arrays. The former maps payload labels to payload values while the latter maps indexing labels to payload labels. The data type's queries are the indexing labels. The specification function

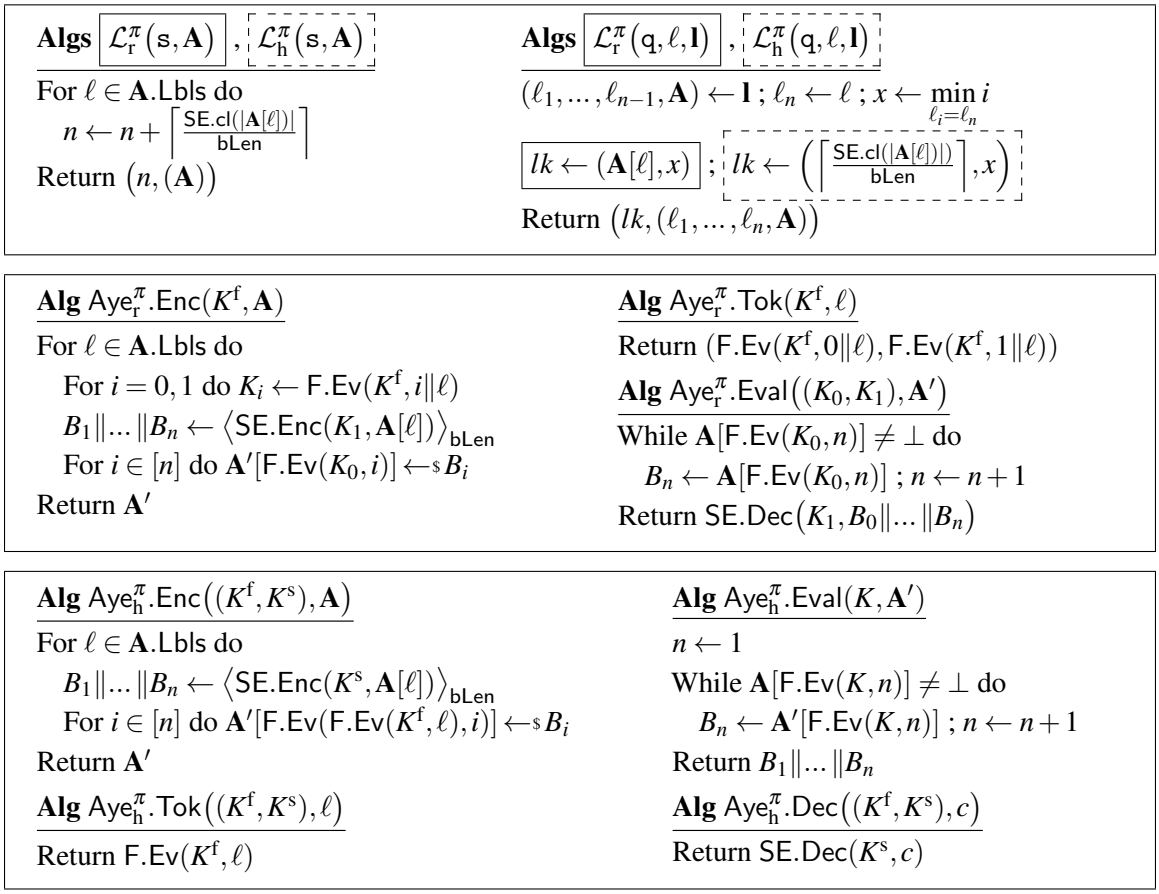

Figure 3.2. Leakage profiles for "standard" RR and RH AYE, and an example of each such scheme.

returns a set of payloads. More specifically:

$$\mathsf{DLdt.Dom} = \{(\mathbf{P},\mathbf{I}) : \mathbf{P},\mathbf{I} \in \mathsf{Adt.Dom}, S \subseteq \mathbf{P}.\mathsf{Lbls} \text{ for all } S \in \mathbf{I}.\mathsf{Vals}\},$$

$$\mathsf{DLdt.QS} = \{0,1\}^* \text{ and } \mathsf{DLdt.Spec}\big((\mathbf{P},\mathbf{I}),\ell\big) = \begin{cases} \{\mathbf{P}[\ell'] : \ell' \in \mathbf{I}[\ell]\} & \text{if } \mathbf{I}[\ell] \neq \perp \\ \perp & \text{otherwise.} \end{cases}$$

Naïve StE for DLdt.

One natural approach to StE for DLdt is to construct an array which maps the indexing labels to the payload values directly (i.e. $\mathbf{A}[\ell] = \{\mathbf{P}[\ell'] : \ell' \in \mathbf{I}[\ell]\}$), then encrypting it with an RH AYE scheme.

The scheme sketched above is correct and secure (with respect to an intuitive leakage profile derived from AYE's) but is space inefficient whenever most payload values exceed their labels in length and are referenced more than once in **I**'s sets. This is unfortunately true in all the applications we discussed in Section 3.1. For example, in encrypted SQL application the payloads are rows in the database and instead of storing pointers to rows in, say, a join in **I** we are instead storing the entire joined relation in **P** (which often contains many duplicate rows).

3.4 "Composite" Double-Level Indexing

In this section, we review the "intuitive" composition technique and highlight how inconsistent simulators create a problem in proving it secure. We then provide a sufficient condition – content oblivious leakage algorithms – on the indexing AYE to recover this proof approach.

"Composite" approach.

We refer to the technique used in the literature to construct StE for DLdt as the *composite approach*. The name stems from the use of one AYE scheme per array – an RH scheme for **P**

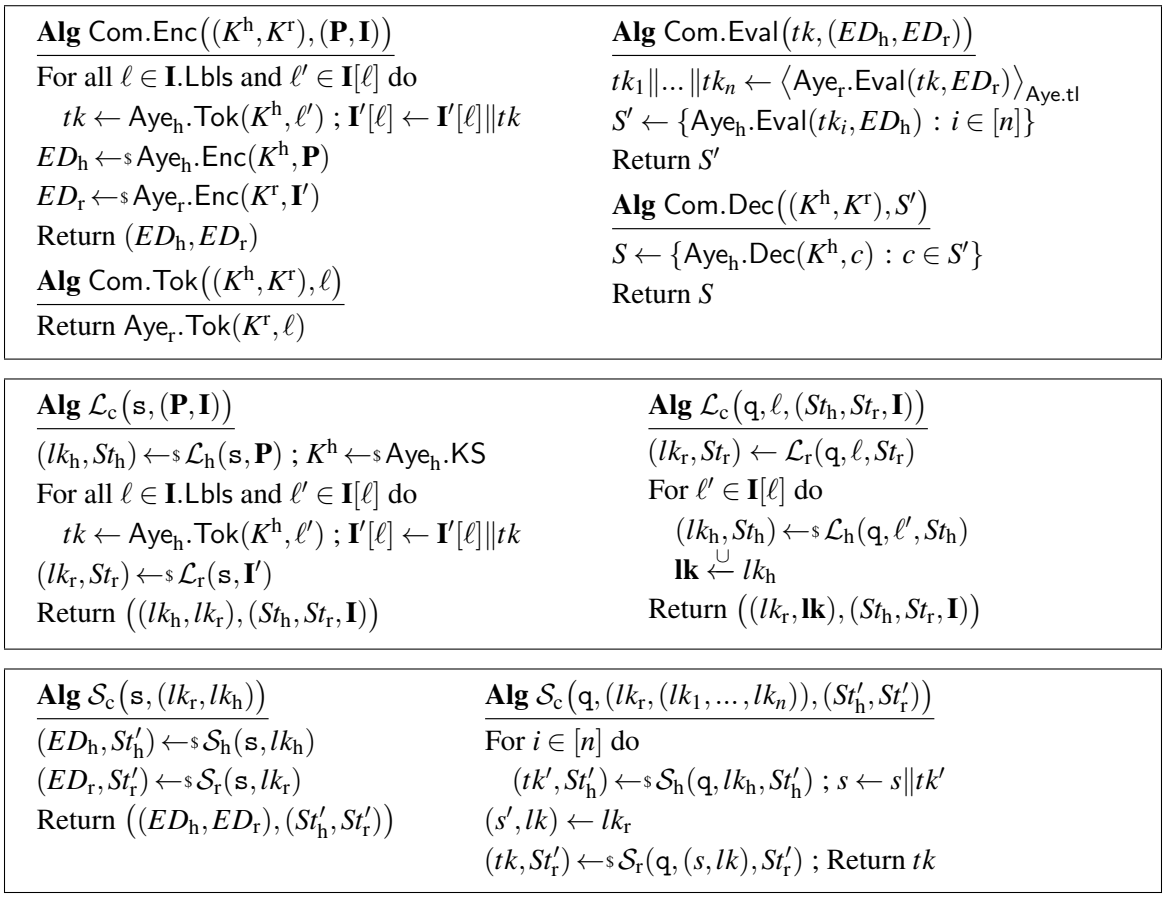

Figure 3.3. Algorithms (left), leakage algorithm (middle) and simulator (right) for "composite" StE scheme $\mathsf{Com} = \mathbf{ComT}[\mathsf{Aye}_\mathrm{h}, \mathsf{Aye}_\mathrm{r}]$ for DLdt. Here, $\mathsf{Aye}_\mathrm{h}, \mathsf{Aye}_\mathrm{r}$ are RH and RR AYEs respectively with leakage algorithms and simulators $\mathcal{L}_\mathrm{h}, \mathcal{S}_\mathrm{h}, \mathcal{L}_\mathrm{r}, \mathcal{S}_\mathrm{r}$. Com is secure if Aye_r has content oblivious leakage.

and a RR scheme for $\mathbf{I}$. Intuitively, this scheme replaces the payload labels $\ell' \in \mathbf{I}[\ell]$ with their respective AYE token, then encrypts $\mathbf{P}, \mathbf{I}$ with the primitives. This technique is reminiscent of those used in SPX, OPX and LabGraph [89, 92, 52].

Now on to the details. The scheme sketched above is derived via a transform with $\mathsf{Com} = \mathbf{ComT}[\mathsf{Aye}_\mathrm{h}, \mathsf{Aye}_\mathrm{r}]$ where $\mathsf{Aye}_\mathrm{h}, \mathsf{Aye}_\mathrm{r}$ are AYE schemes that are RH and RR respectively. It's key set is the product of the primitives' (i.e. $\mathsf{Com.KS} = \mathsf{Aye}_\mathrm{h}.\mathsf{KS} \times \mathsf{Aye}_\mathrm{r}.\mathsf{KS}$). The algorithms for Com are given in Fig. 3.3.

Com**'s security.**

Intuitively, one would expect to be able to compose the leakage algorithms for $\mathsf{Aye}_\mathrm{h}, \mathsf{Aye}_\mathrm{r}$ in some straightforward way to derive a leakage algorithm under which Com can be proven secure. While this is true when $\mathsf{Aye}_\mathrm{h}, \mathsf{Aye}_\mathrm{r}$ have the "standard" leakage profile (see Section 3.3), we demonstrate that pathological leakage prevent this intuition from working in full generality.

The proof issue boils down to the problem of composing the leakage algorithms and simulators of $\mathsf{Aye}_\mathrm{h}, \mathsf{Aye}_\mathrm{r}$ to get one for Com. In Fig. 3.3 we give a composition inspired by those used in prior work[3] Intuitively, $\mathcal{L}_\mathrm{c}\big(\mathtt{s}, (\mathbf{P}, \mathbf{I})\big)$ will construct $\mathbf{I}'$ in the same way $\mathsf{Com.Enc}$ (using a random K^h) then return the setup leakage of $\mathbf{P}, \mathbf{I}'$ as computed by the respective schemes. $\mathcal{L}_\mathrm{c}\big(\mathsf{q}, \ell, St\big)$ will return the leakage associated to querying Aye_r with ℓ, and querying Aye_h with the $\ell' \in \mathbf{I}[\ell]$.

Meanwhile, $\mathcal{S}_\mathrm{c}$'s algorithms channel their inputs into the primitives' and compose their outputs. Recall that because Aye_r is response-revealing, lk_r in $\mathcal{L}_\mathrm{c}, \mathcal{S}_\mathrm{c}$'s query algorithms takes the form $(\mathbf{I}'[\ell], lk)$ for some lk where $\mathbf{I}'$ is as constructed in $\mathcal{L}_\mathrm{c}\big(\mathtt{s}, (\mathbf{P}, \mathbf{I})\big)$. Before this leakage is passed to $\mathcal{S}_\mathrm{r}$, the first argument is replaced with s – the tokens returned by $\mathcal{S}_\mathrm{h}$. This is done so that $\mathsf{Aye}_\mathrm{r}.\mathsf{Eval}(tk, ED_\mathrm{r})$ returns tokens tk' for which $\mathsf{Aye}_\mathrm{h}.\mathsf{Eval}(tk', ED_\mathrm{h}) \neq \perp$. This switch is necessary because the tokens in $\mathbf{I}'[\ell]$ are generated with K^h selected in the leakage algorithm. $\mathcal{S}_\mathrm{h}$ has no knowledge of this key so we can expect that these tokens are unlikely to "work" with the simulated ED_h. However, this switch also means that the behavior of $\mathcal{S}_\mathrm{r}$ in $\mathcal{S}_\mathrm{c}$ is no longer well defined because Aye_r's semantic security only promises that $\mathcal{S}_\mathrm{r}(\mathsf{q}, (\mathbf{I}'[\ell], lk'), St'_\mathrm{r})$ returns a token, not $\mathcal{S}_\mathrm{r}(\mathsf{q}, (s, lk'), St'_\mathrm{r})$.

Content oblivious leakage.

We say that an RR AYE scheme has *content oblivious leakage* if homomorphic arrays have the same leakage (modulo query responses). Let $\mathbf{A}_1, \mathbf{A}_2 \in \mathsf{Adt.Dom}$ be homomorphic (as defined in Section 3.3), $\ell_1, \ldots, \ell_n \in \{0,1\}^*$ be labels and Aye_r be a RR AYE scheme with leakage

[3] An alternative approach to $\mathcal{L}_\mathrm{c}$ is to generate the tokens in $\mathbf{I}'$ using $\mathcal{S}_\mathrm{r}$. We note that even with this leakage profile, the proof fails for similar reasons.

algorithm $\mathcal{L}$. Then, we say that $\mathcal{L}$ is content oblivious if there exists $lk_0, lk_1, \ldots, lk_n$ such that for $i = 1, 2$ and all random coins:

$$
\begin{aligned}
(lk_0, St_i) &\leftarrow\!\!\$\ \mathcal{L}(\mathtt{s}, \mathbf{A}_i) \\
\big((\mathbf{A}_i[\ell_1], lk_1), St_i\big) &\leftarrow\!\!\$\ \mathcal{L}(\mathtt{q}, \ell_1, St_i) \\
\vdots \qquad & \qquad \vdots \\
\big((\mathbf{A}_i[\ell_n], lk_n), St_i\big) &\leftarrow\!\!\$\ \mathcal{L}(\mathtt{q}, \ell_n, St_i).
\end{aligned}
$$

Query obliviousness is sufficient to ensure that Com is semantically secure via the proof sketched above because it would guarantee that the substitution of s' with s in the input to $\mathcal{S}_\mathrm{r}$ will return a token which can be evaluated to the desired result. This gives us:

Theorem 18 *Let* Aye_r *be an RR AYE with content oblivious leakage and* Aye_h *be an RH AYE. Then, given adversary* $\mathcal{A}$*, leakage algorithms* $\mathcal{L}_\mathrm{r}, \mathcal{L}_\mathrm{h}$ *and simulators* $\mathcal{S}_\mathrm{r}, \mathcal{S}_\mathrm{h}$*, one can construct* $\mathcal{A}_\mathrm{r}, \mathcal{A}_\mathrm{h}, \mathcal{S}_\mathrm{c}$ *such that*

$$
\mathbf{Adv}^{\mathrm{SS}}_{\mathsf{Com},\mathcal{L}_\mathrm{c},\mathcal{S}_\mathrm{c}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{SS}}_{\mathsf{Aye}_\mathrm{h},\mathcal{L}_\mathrm{h},\mathcal{S}_\mathrm{h}}(\mathcal{A}_\mathrm{h}) + \mathbf{Adv}^{\mathrm{SS}}_{\mathsf{Aye}_\mathrm{r},\mathcal{L}_\mathrm{r},\mathcal{S}_\mathrm{r}}(\mathcal{A}_\mathrm{r})
$$

where $\mathsf{Com} = \mathbf{ComT}[\mathsf{Aye}_\mathrm{h}, \mathsf{Aye}_\mathrm{r}], \mathcal{L}_\mathrm{c}$ *are as described in Fig. 3.3.*

Note that $\mathsf{Aye}_\mathrm{r}^{\pi}$ (in Fig. 3.2) has content oblivious leakage. Its setup leakage is the number of blocks in $\mathbf{A}'$ which is constant for homomorphic arrays because SE's ciphertext length function is message independent. The query leakage (apart from the query response) is the query equality pattern (which is independent of $\mathbf{A}$).

Counterexample.

To complete our analysis, we give a leakage algorithm $\mathcal{L}_\mathrm{r}$ and simulator $\mathcal{S}_\mathrm{r}$ for Aye_r which renders $\mathsf{Com} = \mathbf{ComT}[\mathsf{Aye}_\mathrm{h}, \mathsf{Aye}_\mathrm{r}]$ insecure with respect to the leakage algorithm $\mathcal{L}_\mathrm{c}$ which calls $\mathcal{L}_\mathrm{r}$.

Given Aye_r and leakage algorithms $\mathcal{L}, \mathcal{S}$, define $\mathcal{L}_\mathrm{r}, \mathcal{S}_\mathrm{r}$ to work as follows:

$$\mathcal{L}_\mathrm{r}(\mathtt{s}, \mathsf{Aye}_\mathrm{r}) = \mathcal{L}(\mathtt{s}, \mathsf{Aye}_\mathrm{r}) \text{ and } \mathcal{S}_\mathrm{r}(\mathtt{s}, lk) = \mathcal{S}(\mathtt{s}, lk)$$

$$\mathcal{L}_\mathrm{r}(\mathtt{q}, \ell, St) = (s, (s, lk)) \text{ where } \mathcal{L}(\mathtt{q}, \ell, St) = (s, lk)$$

$$\mathcal{S}_\mathrm{r}(\mathtt{q}, (s_1, (s_2, lk)), St') = \begin{cases} \mathcal{S}(\mathtt{q}, (s_1, lk), St') & \text{if } s_1 = s_2 \\ \\ \perp & \text{otherwise.} \end{cases}$$

Notice that because the second part of the query leakage contains s, the query output, it is not content oblivious. Further, notice that Aye_r security under $\mathcal{L}_\mathrm{r}, \mathcal{S}_\mathrm{r}$ means it is also secure with respect to $\mathcal{L}, \mathcal{S}$ because the simulator in $\mathrm{G}^{\mathrm{ss}}_{\mathsf{Aye}_\mathrm{r}, \mathcal{L}_\mathrm{r}, \mathcal{S}_\mathrm{r}}$ is always given input with $s_1 = s_2$.

At the same time, when used in $\mathcal{L}_\mathrm{c}, \mathcal{S}_\mathrm{c}$, simulator $\mathcal{S}_\mathrm{r}$ will (with high probability) return $\perp$ whenever it is asked to simulate token generation because the two s_i presented to $\mathcal{S}_\mathrm{r}$ in $\mathrm{G}^{\mathrm{ss}}_{\mathsf{Com}, \mathcal{L}_\mathrm{c}, \mathcal{S}_\mathrm{c}}$ are strings of tokens generated with different Aye_h keys and are unlikely to be equal (for "standard" choices of Aye_h). This never happens in the "real" world, so the adversary that requests a valid token then returns 0 (i.e. guesses that it is in the simulated world) whenever $tk = \perp$ has high advantage.

It is worth noting the simulator $\mathcal{S}_\mathrm{r}$ presented here is not the only simulator which satisfies the security definition of a response revealing data structure (for example $\mathcal{S}$ also works). This illustrates that the issue is not necessarily an issue of the leakage function. The leakage function presented here *could* work in the proof, so long as the proof isn't using a "misbehaving" simulator like $\mathcal{S}_\mathrm{r}$. This observation allows us to solve the "bug" in prior proofs with a weaker assumption than content oblivious leakage. Specifically, we could change the security definition for response revealing data structures to stating a formalization of "there must exist a simulator that doesn't misbehave on inputs modified in this way." After this modification, we could prove security for even non-content oblivious leakage and use these simulators in composition. However, all simulators for structures which are secure under content oblivious leakage cannot misbehave

on inputs of the form above, and "standard" structured encryption is already secure under the content oblivious leakage definition. So, we make the decision to explicitly focus on the sufficient leakage assumption rather than introduce a modified security definition.

We leave open the problem of classifying structured encryption schemes, which can have misbehaving simulators, and classifying schemes where all simulators satisfying the security definition must misbehave. Since, standard schemes are already content oblivious, this is not a pressing problem, but one of potential theoretical interest.

3.4.1 Inconsistent simulators in prior work.

The leakage algorithms and security proofs of SPX, OPX and LabGraph in [89, 92, 52] follow a similar technique to the one discussed above. However, they make insufficient assumptions about the underlying component schemes, leakage algorithms or simulators meaning that their proofs are "buggy". Assuming content oblivious leakage is a sufficient condition to resolve the issues in their proofs though, meaning that their security results still hold when standard primitives are used. Note that in our discussions below, our references (to sections, appendices, definitions, theorems) and notation follow that of the papers in question.

Bug in SPX/OPX.

Although the bug does occur multiple times in SPX and OPX [89, 92], we will only outline a single occurrence for brevity. The other occurrences can be fixed using the same content oblivious assumption. We note that this issue is not resolved by the authors assumptions on their AYE primitives[4].

The relevant definition for the following error is Definition 4.3 at the end of Section 4 in SPX. In the ideal game of this definition, the simulator only receives inputs of the form $(DS(q_i), \mathcal{L}_{\mathsf{Q}}(DS, q_i))$ to generate tokens.

We observe in the actual proof, the simulator $\mathcal{S}_{\mathsf{MM}}$ is not fed inputs of the same form

[4]In particular, their assumption that one of the primitives in OPX be Σ^{π} (an AYE very similar to $\mathsf{Aye}_{\mathsf{h}}^{\pi}$).

as in the security definition. The occurrence we focus on is in section "Appendix F: Proof of Theorem 6.1" of SPX. The base simulator $\mathcal{S}_{\mathsf{MM}}$ is the simulator which exists based on the security Definition 4.3.

In describing the simulator, the authors write,

$$\mathsf{rtk}_{\mathbf{r}} \leftarrow \mathcal{S}_{\mathsf{MM}}((\mathsf{ct}_j)_{j\in[\#\mathbf{r}]}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}(\mathsf{MM}_R, \chi(\mathbf{r})))$$

And, then in the proof pass these $\mathsf{rtk}_{\mathbf{r}}$ into another simulator,

$$\mathbf{tk}_{i,j} \leftarrow \mathcal{S}_{\mathsf{MM}}((\mathsf{rtk}_{\mathbf{r}})_{\mathbf{r}\in\mathsf{DB}_{\mathsf{att}_{i,j}=x_{i,j}}}, \mathcal{L}^{\mathsf{mm}}_{\mathsf{Q}}(\mathsf{MM}_V, \chi(\mathsf{att}_{i,j})))$$

However, there is no guarantee that this input to $\mathcal{S}_{\mathsf{MM}}$ fits the input form required by Definition 4.3, because $(\mathsf{rtk}_{\mathbf{r}})_{\mathbf{r}\in\mathsf{DB}_{\mathsf{att}_{i,j}=x_{i,j}}}$ (tokens for MM_R) are not necessarily the same tokens contained in $\mathsf{MM}_V(\chi(\mathsf{att}_{i,j}))$. Unless the leakage from the generation of the simulated $\mathsf{rtk}_{\mathbf{r}}$ leaks the tokens in MM_R or a way to generate them, then it is unlikely over a random key choice the $\mathsf{rtk}_{\mathbf{r}}$ generated correspond to the the values stored in MM_V.

Note, this is not a necessary behavior of $\mathcal{S}_{\mathsf{MM}}$, but one that is not ruled out. Either the content obliviousness assumption or modified security definition illustrated above would avoid this issue and allow the proof to go through.

Bug in LabGraph.

In LabGraph, the same bug is made in the proof of Theorem 6.2 [52]. At the beginning of the proof, the authors outline a simulator $\mathcal{S}$. In step 2b, $\mathcal{S}$ feeds in $\mathbf{v}_w$ generated from other simulators to generator a token τ_w. However, these simulators may not necessarily generate stored tokens τ^+ and τ^- with high probability.

It is worth noting these authors require their structured encryption algorithms to be "chainable," which places restrictions on both the setup and query leakage. However, the security definition (Definition 4.2) indicates the simulator for queries will receive input of the form

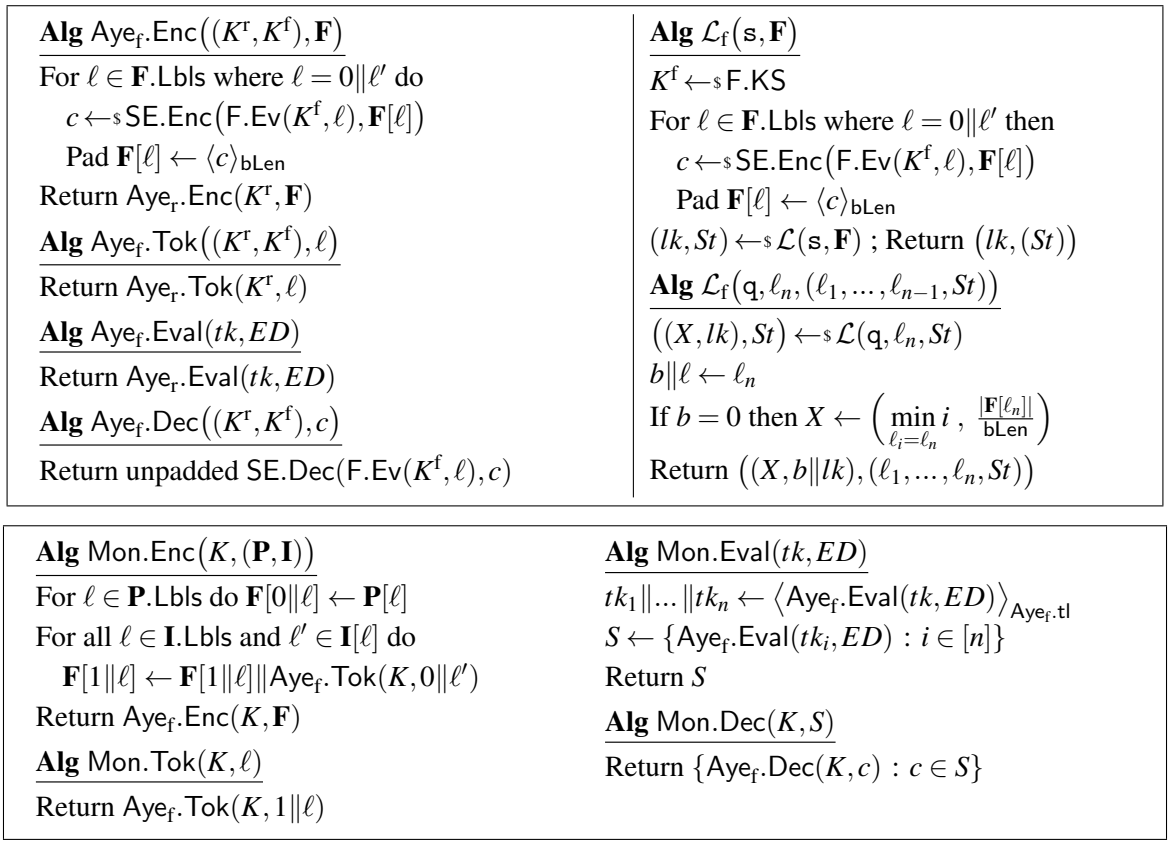

Figure 3.4. Top: Algorithms (left) and leakage profile (right) for RF AYE scheme $\mathsf{Aye_f} = \mathbf{RfT}[\mathsf{Aye_r},\mathsf{SE},\mathsf{F}]$ where $\mathsf{Aye_r},\mathsf{SE},\mathsf{F}$ is a RR AYE scheme (with leakage algorithm $\mathcal{L}$), a symmetric encryption scheme and a function family respectively. Bottom: "monolithic" StE scheme $\mathsf{Mon} = \mathbf{MonT}[\mathsf{Aye_f}]$.

$(\mathcal{L}_2(\delta,q),\mathbf{v}_I)$ where $I := \mathsf{Query}(\delta,q)$. But, the restrictions on the query leakage function in their Definition 6.1 (Chainability) do not rule out the existence of a simulator fitting the security definition but which behaves in an unspecified way on mismatched input. As with SPX and OPX, either our sufficient assumption or a modified security definition would eliminate this proof bug.

3.5 "Monolithic" Double-Level Indexing via Key-Dependent AYE

Now we give an approach to StE DLdt that differs from those used in the literature which we call the *monolithic* approach. Evaluating its security requires a new KD-security notion

which allows the adversary to compute and store tokens in their array data structures. We show that many AYE schemes from the literature achieve this notion of security and argue that the monolithic approach is superior to the composite one from Section 3.4.

Response-flexible StE.

We start by formalizing a third response type for StE schemes then describe AYE can achieve it. Intuitively, a response flexible (RF) StE scheme allows the client to indicate their desired response type (RH or RR) using the first bit of the query. More formally, if StE for DT is RF, then if DS, q, tk, ED are as defined in StE's syntax then $\mathsf{StE.Eval}(tk, ED) = \mathsf{DT.Spec}(DS, q')$ whenever $q = 1\|q'$. We will assume that $\mathsf{StE.Dec}$ is only called when queries of the form $q = 0\|q'$ are made.

In our monolithic scheme, we need a RF AYE, so we give an example of such a scheme $\mathsf{Aye_f}$ which is defined using transform **RfT**. Let SE, F be a symmetric encryption scheme and function family such that $\{0,1\}^{\mathsf{F.ol}} = \mathsf{SE.KS}$. Intuitively, when $\ell = 0\|\ell'$, $\mathsf{Aye_f}$ "hides" $\mathbf{F}[\ell]$ by symmetrically encrypting it under a key derived using ℓ and F. We define $\mathsf{Aye_f.KS} = \mathsf{Aye_r.KS} \times \mathsf{F.KS}$ and detail its algorithms in Fig. 3.4. Later, we prove its security under a KD-security notion with respect to the leakage algorithm $\mathcal{L}_{\mathrm{f}}$ in Fig. 3.4.

Monolithic solution.

Armed with the above new primitive, a natural way to improve the composite solution is to merge the contents of $\mathbf{P}$ and $\mathbf{I}'$ into a single array $\mathbf{F}$ then encrypt this using an RF AYE scheme with labels that are one bit longer. We refer to this as the "monolithic approach" to StE for DLdt since a single monolithic data structure is used for both payload storage and indexing. The scheme Mon is constructed via transform **MonT** which takes as input a RF AYE scheme $\mathsf{Aye_f}$. Mon's key set is $\mathsf{Aye_f.KS}$ and its algorithms are depicted in Fig. 3.4. We note that, using standard primitives such as those in Fig. 3.2, it is important that $\mathbf{F}$'s values are padded to a multiple of bLen, then encrypted, then padded again to avoid padding oracle attacks.

Game $\mathrm{G}^{\mathcal{F}\text{-ss}}_{\mathsf{StE},\mathcal{L},\mathcal{S}}(\mathcal{A})$

$K \leftarrow\!\!\$\ \mathsf{StE.KS}$; $b \leftarrow\!\!\$\ \{0,1\}$
$(f_0, St_{\mathrm{a}}) \leftarrow\!\!\$\ \mathcal{A}(\mathtt{s})$
If $b = 1$ then
  $ED \leftarrow\!\!\$\ \mathsf{StE.Enc}(K, f_0(K))$
Else
  $(lk_0, St) \leftarrow\!\!\$\ \mathcal{L}(\mathtt{s}, DS)$
  $(ED, St') \leftarrow\!\!\$\ \mathcal{S}(\mathtt{s}, lk_0)$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\textsc{Tok}}(\mathtt{q}, ED, St_{\mathrm{a}})$
Require $(f_0, f_1, \ldots, f_n) \in \mathcal{F}$
Return $(b = b')$

Oracle $\textsc{Tok}(f)$

$n \leftarrow n+1$; $f_n \leftarrow f$
If $b = 1$ then
  $tk_i \leftarrow\!\!\$\ \mathsf{StE.Tok}(K, f_n(K))$
Else
  $(lk_n, St) \leftarrow\!\!\$\ \mathcal{L}(\mathtt{q}, f_n(K), St)$
  $(tk_n, St') \leftarrow\!\!\$\ \mathcal{S}(\mathtt{q}, lk_n, St')$
Return tk_n

Alg $\mathcal{L}_{\mathrm{m}}\big(\mathtt{s}, (\mathbf{P}, \mathbf{I})\big)$

$K \leftarrow\!\!\$\ \mathsf{Aye}_{\mathrm{r}}.\mathsf{KS}$
For $\ell \in \mathbf{P}.\mathsf{Lbls}$ do $\mathbf{F}[0\|\ell] \leftarrow \mathbf{P}[\ell]$
For all $\ell \in \mathbf{I}.\mathsf{Lbls}$ and $\ell' \in \mathbf{I}[\ell]$ do
  $\mathbf{F}[1\|\ell] \leftarrow \mathbf{F}[1\|\ell] \| \mathsf{Aye}_{\mathrm{f}}.\mathsf{Tok}(K, 0\|\ell')$
$(lk, St) \leftarrow\!\!\$\ \mathcal{L}_{\mathrm{f}}\big(\mathtt{s}, \mathbf{F}\big)$; Return $\big(lk, (St, \mathbf{I})\big)$

Alg $\mathcal{L}_{\mathrm{m}}\big(\mathtt{q}, \ell, (St, \mathbf{I})\big)$

$(lk, St) \leftarrow \mathcal{L}_{\mathrm{f}}\big(\mathtt{q}, 1\|\ell, St\big)$
For $\ell' \in \mathbf{I}[\ell]$ do
  $(lk', St) \leftarrow\!\!\$\ \mathcal{L}_{\mathrm{r}}(\mathtt{q}, 0\|\ell', St)$; $\mathbf{lk} \xleftarrow{\cup} lk'$
Return $\big((lk, \mathbf{lk}), (St, \mathbf{I})\big)$

Alg $\mathcal{S}_{\mathrm{m}}\big(\mathtt{s}, lk\big)$

$(ED, St') \leftarrow\!\!\$\ \mathcal{S}_{\mathrm{f}}\big(\mathtt{s}, lk\big)$; Return (ED, St')

Alg $\mathcal{S}_{\mathrm{m}}\big(\mathtt{q}, ((S', lk), \mathbf{lk}), St'\big)$

For $lk' \in \mathbf{lk}$ do
  $(tk', St') \leftarrow\!\!\$\ \mathcal{S}_{\mathrm{f}}\big(\mathtt{q}, lk_{\mathrm{r}}, St'\big)$; $S \xleftarrow{\cup} tk'$
$(tk, St') \leftarrow\!\!\$\ \mathcal{S}_{\mathrm{f}}\big(\mathtt{q}, (S, lk), St'\big)$; Return tk

Figure 3.5. Left: Game defining adaptive $\mathcal{F}$-semantic security of StE for function class $\mathcal{F}$ with respect to $\mathcal{L}, \mathcal{S}, \mathcal{A}$. Right: Leakage algorithm (top) and simulator (bottom) for "monolithic" StE scheme $\mathsf{Mon} = \mathbf{MonT}[\mathsf{Aye}_{\mathrm{f}}]$ where $\mathsf{Aye}_{\mathrm{f}}$ is an RF AYE scheme (with respect to $\mathcal{L}_{\mathrm{f}}, \mathcal{S}_{\mathrm{f}}$).

Intuitively, Mon should be superior to Com under standard primitives because the monolithic array has lower setup leakage. In particular, using Com the adversary will learn the number of blocks in two arrays $(\mathbf{I}', \mathbf{P})$ while with Mon it learns the number of blocks in only one $(\mathbf{F})$. However, because $\mathbf{F}$ stores tokens to its own content we need a game which allows the client to submit submit key-dependent data structures to show that Com is secure. We do this by generalizing our notion of semantic security to Key-Dependent (KD) security below.

KD semantic security.

We define semantic security for StE schemes with respect to a function class $\mathcal{F}$, leakage algorithm $\mathcal{L}$ and simulator $\mathcal{S}$ using the game $\mathrm{G}^{\mathcal{F}\text{-ss}}_{\mathsf{StE},\mathcal{L},\mathcal{S}}$ depicted in Fig. 3.5. This game generalizes the semantic security notion given in Section 3.2 by computing the data structure DS and queries

$q_1, \ldots, q_n$ using adversary-provided (deterministic) functions $f_0, f_1, \ldots, f_n$.

We define the universal function class $\mathcal{F}_{\text{all}}$ for DT to include all $(f_0, f_1, \ldots, f_n)$ where $f_0(K) \in \mathsf{DT.Dom}$ and $f_i(K) \in \mathsf{DT.QS}$ for all $i \in [n]$. We require that $\mathcal{F} \subseteq \mathcal{F}_{\text{all}}$ so that $\mathrm{G}^{\mathcal{F}\text{-ss}}_{\mathsf{StE},\mathcal{L},\mathcal{S}}$ is syntactically sound. We can also recover CK's notion of semantic security by restricting the function class to the set of *key independent functions* $\mathcal{F}_{\text{kInd}}$ which is defined as all $(f_0, f_1, \ldots, f_n) \in \mathcal{F}_{\text{all}}$ where all the f_i are constant functions.

Token generating function class.

We define a function class $\mathcal{F}_{\text{tok}}$ for AYE scheme Aye which allows the adversary to encrypt arrays which depend on AYE tokens. This can be used, for example, to construct arrays whose values are tokens.

If $(f_0, f_1, \ldots, f_n) \in \mathcal{F}_{\text{tok}}$ then $f_1, \ldots, f_n$ are constant functions and $f_0(K) = f_{\text{tok}}^{\mathsf{Aye.Tok}(K,\cdot)}$ where f_{tok} is a function that generates DS with access to a token-generating oracle.

Constructing $\mathcal{F}_{\text{tok}}$-secure AYE.

We show that many mainstream AYE schemes are $\mathcal{F}_{\text{tok}}$-secure then discuss how these can be used to construct $\mathcal{F}_{\text{tok}}$-secure RF AYE schemes using **RfT**. We observed that many AYE schemes (such as those in [48, 56]) sselect a function family key K^{t} and use this exclusively for generating tokens. We demonstrate that such schemes are $\mathcal{F}_{\text{tok}}$-secure.

If Aye is a *PRF-based AYE* scheme (of any response type), then there exists function family F, key set KS, and algorithms $\mathsf{Enc}, \mathsf{Tok}$ such that $\mathsf{Aye.KS} = \mathsf{F.KS} \times \mathsf{KS}$ and:

$$\mathsf{Aye.Enc}\big((K^{\mathsf{t}}, K), DS\big) = \mathsf{Enc}^{\mathsf{F.Ev}(K^{\mathsf{t}},\cdot)}(K, DS),$$
$$\mathsf{Aye.Tok}\big((K^{\mathsf{t}}, K), \ell\big) = \mathsf{Tok}^{\mathsf{F.Ev}(K^{\mathsf{t}},\cdot)}(\ell).$$

Then, Aye's $\mathcal{F}_{\text{kInd}}$-security (i.e. semantic security as defined in Section 3.2) implies its $\mathcal{F}_{\text{tok}}$-security because $\mathsf{F.Ev}(K^{\mathsf{t}}, \cdot)$ is indistinguishable from a (key-independent) random function $\phi : \{0,1\}^* \to \{0,1\}^{\mathsf{F.ol}}$ (assuming F is a PRF).

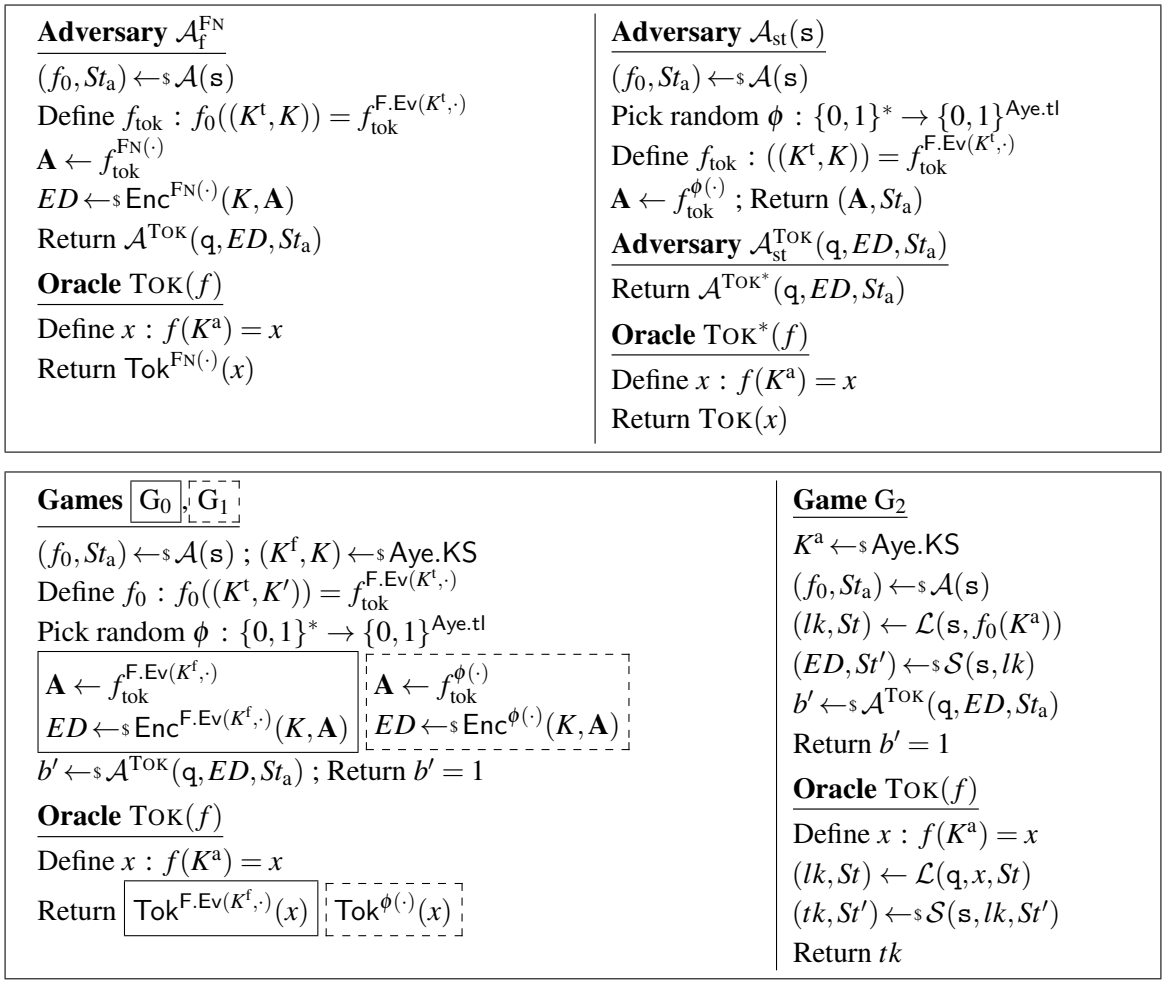

Figure 3.6. Adversaries and games used in proof of Theorem 19. Here, $\mathsf{Enc}, \mathsf{Tok}$ are the algorithms used in the definition of Aye as a PRF-based scheme.

Theorem 19 *Let* Aye *be a PRF-based AYE scheme. Then, given adversary* $\mathcal{A}$*, leakage algorithm* $\mathcal{L}$ *and simulator* $\mathcal{S}$*, one can construct* $\mathcal{A}_{\mathrm{f}}, \mathcal{A}_{\mathrm{st}}$ *such that*

$$\mathbf{Adv}^{\mathcal{F}_{\mathrm{tok}}\text{-}\mathrm{ss}}_{\mathsf{Aye},\mathcal{L},\mathcal{S}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{A}_{\mathrm{f}}) + \mathbf{Adv}^{\mathrm{ss}}_{\mathsf{Aye},\mathcal{L},\mathcal{S}}(\mathcal{A}_{\mathrm{ss}}).$$

Proof. We define adversaries $\mathcal{A}_{\mathrm{st}}, \mathcal{A}_{\mathrm{f}}$ as in Fig. 3.6. By the definition of $\mathcal{F}_{\mathrm{tok}}$ we can express f_0 as f_{tok} (with oracle access to the PRF) and assume the query phase functions are constant. Additionally, let $\mathsf{Enc}, \mathsf{Tok}$ be the algorithms used in the definition of Aye as a PRF-based scheme. Both adversaries run $\mathcal{A}$, with $\mathcal{A}_{\mathrm{st}}$ simulating the PRF using a random mapping ϕ and $\mathcal{A}_{\mathrm{f}}$ simulating the whole $\mathcal{F}_{\mathrm{tok}}$-security game but using its FN oracle whenever the PRF is required. Notice that $\mathcal{A}_{\mathrm{st}}$ constructs Aye without using the key and therefore can play the $\mathrm{G}^{\mathrm{ss}}_{\mathsf{Aye},\mathcal{L},\mathcal{S}}$ game.

Now we can conclude via a standard hybrid argument using games $\mathrm{G}_0, \mathrm{G}_1, \mathrm{G}_2$ depicted in Fig. 3.6. From the definition of $\mathcal{F}_{\mathrm{tok}}$-security, we have $\mathbf{Adv}^{\mathcal{F}_{\mathrm{tok}}\text{-}\mathrm{ss}}_{\mathsf{Aye},\mathcal{L},\mathcal{S}}(\mathcal{A}) = \Pr[\mathrm{G}_0] - \Pr[\mathrm{G}_2]$. From the definitions of PRF-security and $\mathcal{A}_{\mathrm{f}}$ we have $\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{A}_{\mathrm{f}}) = \Pr[\mathrm{G}_0] - \Pr[\mathrm{G}_1]$. Finally, from the definitions of semantic security for Aye and $\mathcal{A}_{\mathrm{st}}$ we have $\mathbf{Adv}^{\mathrm{ss}}_{\mathsf{Aye},\mathcal{L},\mathcal{S}}(\mathcal{A}_{\mathrm{st}}) = \Pr[\mathrm{G}_1] - \Pr[\mathrm{G}_2]$. Combining these equations gives the desired result. □

RfT preserves $\mathcal{F}$-security.

The leakage profile of $\mathsf{Aye}_{\mathrm{f}} = \mathbf{RfT}[\mathsf{Aye}_{\mathrm{r}}, \mathsf{SE}, \mathsf{F}]$ can be constructed from that of $\mathsf{Aye}_{\mathrm{r}}$ as depicted in Fig. 3.4. Intuitively, this profile is identical to $\mathsf{Aye}_{\mathrm{r}}$'s but its query leakage includes the response type, query pattern, number of blocks returned.

RfT preserves semantic security under any function class. This means that if $\mathsf{Aye}_{\mathrm{r}}$ is a PRF-based AYE the resultant $\mathsf{Aye}_{\mathrm{f}}$ is sufficiently secure to be used in our monolithic solution.

Theorem 20 *Given adversary* $\mathcal{A}$*, leakage algorithm* $\mathcal{L}$ *and function class* $\mathcal{F}$*, let* $\mathsf{Aye}_{\mathrm{r}}$ *be a RR AYE scheme and* $\mathsf{Aye}_{\mathrm{f}} = \mathbf{RfT}[\mathsf{Aye}_{\mathrm{r}}, \mathsf{SE}, \mathsf{F}], \mathcal{L}_{\mathrm{f}}$ *be as defined in Fig. 3.4. Then, given simulator* $\mathcal{S}$*,*

Alg $\mathcal{S}_{\mathrm{f}}(\mathtt{s}, lk)$
$(lk, St') \leftarrow\!\$\ \mathcal{S}(\mathtt{s}, lk)$
Return $\big(lk, (St', \mathbf{V})\big)$

Alg $\mathcal{S}_{\mathrm{f}}\big(\mathtt{q}, (X, b\|lk), (St', \mathbf{V})\big)$
If $b = 1$ then $(tk, St') \leftarrow\!\$\ \mathcal{S}(\mathtt{q}, (X, lk), St')$
Else
  $(i, s) \leftarrow X$
  If $\mathbf{V}[i] = \bot$ then $\mathbf{V}[i] \leftarrow\!\$\ \{0,1\}^{s \cdot \mathsf{bLen}}$
  $(tk, St') \leftarrow\!\$\ \mathcal{S}(\mathtt{q}, (\mathbf{V}[i], lk), St')$
Return $\big(tk, (St', \mathbf{V})\big)$

Adversary $\mathcal{A}_{\mathrm{ss}}(\mathtt{s})$
$(f_0, St_{\mathrm{a}}) \leftarrow\!\$\ \mathcal{A}(\mathtt{s})$; $K^{\mathrm{f}} \leftarrow\!\$\ \mathsf{F.KS}$
Return (f_0^*, St_{a})

Function $f_0^*(K^{\mathrm{r}})$
$\mathbf{A} \leftarrow f_0(K^{\mathrm{r}})$
For $\ell \in \mathbf{A}.\mathsf{Lbls}$ where $\ell = 0\|\ell'$ do
  $\mathbf{A}[\ell] \leftarrow\!\$\ \mathsf{SE.Enc}(\mathsf{F.Ev}(K^{\mathrm{f}}, \ell), \mathbf{A}[\ell])$
Return $\mathbf{A}$

Adversary $\mathcal{A}_{\mathrm{ss}}(\mathtt{q}, St_{\mathrm{a}})$
Return $\mathcal{A}(\mathtt{q}, St_{\mathrm{a}})$

Adversary $\mathcal{A}_{\mathrm{f}}^{\mathrm{FN}}$
$(f_0, St_{\mathrm{a}}) \leftarrow\!\$\ \mathcal{A}(\mathtt{s})$
$K^{\mathrm{r}} \leftarrow\!\$\ \mathsf{Aye}_{\mathrm{r}}.\mathsf{KS}$
$\mathbf{A} \leftarrow f_0(K^{\mathrm{r}})$
For $\ell \in \mathbf{A}.\mathsf{Lbls}$ do
  If $\ell = 0\|\ell'$ then
    $K \leftarrow\!\$\ \mathrm{FN}(\ell)$
    $\mathbf{A}[\ell] \leftarrow\!\$\ \mathsf{SE.Enc}(K, \mathbf{A}[\ell])$
$(lk, St) \leftarrow\!\$\ \mathcal{L}(\mathtt{s}, \mathbf{A})$
$(ED, St') \leftarrow\!\$\ \mathcal{S}(\mathtt{s}, lk)$
Return $\mathcal{A}^{\mathrm{TOK}}(\mathtt{q}, ED, St_{\mathrm{a}})$

Oracle $\mathrm{TOK}(f)$
$(lk, St) \leftarrow\!\$\ \mathcal{L}(\mathtt{q}, f(K^{\mathrm{r}}), St)$
$(tk, St') \leftarrow\!\$\ \mathcal{S}(\mathtt{s}, lk, St')$
Return tk

Adversary $\mathcal{A}_{\mathrm{se}}^{\mathrm{ENC}}$
$(f_0, St_{\mathrm{a}}) \leftarrow\!\$\ \mathcal{A}(\mathtt{s})$; $K^{\mathrm{r}} \leftarrow\!\$\ \mathsf{Aye}_{\mathrm{r}}.\mathsf{KS}$
$\mathbf{A} \leftarrow f_0(K^{\mathrm{r}})$; $m \leftarrow\!\$\ [M]$
For $\ell \in \mathbf{A}.\mathsf{Lbls}$ where $\ell = 0\|\ell'$ do
  $K \leftarrow\!\$\ \mathsf{SE.KS}$; $\mathbf{A}[\ell] \leftarrow\!\$\ \mathsf{SE.Enc}(K, \mathbf{A}[\ell])$
$(lk, St) \leftarrow\!\$\ \mathcal{L}(\mathtt{s}, \mathbf{A})$; $(ED, St') \leftarrow\!\$\ \mathcal{S}(\mathtt{s}, lk)$
Return $\mathcal{A}^{\mathrm{TOK}}(\mathtt{q}, ED, St_{\mathrm{a}})$

Oracle $\mathrm{TOK}(f)$
$n \leftarrow n + 1$; $q_n \leftarrow f(K^{\mathrm{r}})$
$x \leftarrow \min_{q_i = q_n} i$; $((X, lk), St) \leftarrow\!\$\ \mathcal{L}(\mathtt{q}, q_n, St)$
If $q_n = 0\|\ell$ then
  If $\mathbf{V}[x] = \bot$ then
    If $m < n$ then $\mathbf{V}[x] \leftarrow X$
    If $m = n$ then $\mathbf{V}[x] \leftarrow \mathrm{ENC}(\mathbf{A}[\ell])$
    If $m > n$ then $\mathbf{V}[x] \leftarrow\!\$\ \{0,1\}^{|X|}$
$(tk, St') \leftarrow\!\$\ \mathcal{S}(\mathtt{s}, (\mathbf{V}[x], lk), St')$; Return tk

Figure 3.7. Simulator (top left) and adversaries used in proof of Theorem 20. In $\mathcal{A}_{\mathrm{se}}$, M is the maximum number of queries response-hiding queries $\mathcal{A}$ makes to TOK.

Game G_0

$(f_0, St_{\mathrm{a}}) \leftarrow\!\!\$\ \mathcal{A}(\mathtt{s})$; $K^{\mathrm{r}} \leftarrow\!\!\$\ \mathsf{Aye}_{\mathrm{r}}.\mathsf{KS}$
$K^{\mathrm{f}} \leftarrow\!\!\$\ \mathsf{F}.\mathsf{KS}$; $\mathbf{A} \leftarrow f_0(K^{\mathrm{r}})$
For $\ell \in \mathbf{A}.\mathsf{Lbls}$ where $\ell = 0\|\ell'$ do
  $K \leftarrow \mathsf{F}.\mathsf{Ev}(K^{\mathrm{f}}, \ell)$
  $\mathbf{A}[\ell] \leftarrow\!\!\$\ \mathsf{SE}.\mathsf{Enc}(K, \mathbf{A}[\ell])$
$ED \leftarrow\!\!\$\ \mathsf{Aye}_{\mathrm{r}}.\mathsf{Enc}(K^{\mathrm{r}}, \mathbf{A})$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{Tok}}(\mathtt{q}, ED, St_{\mathrm{a}})$
Return $b' = 1$

Oracle $\mathrm{Tok}(f)$

Return $\mathsf{Aye}_{\mathrm{r}}.\mathsf{Tok}(K^{\mathrm{r}}, f(K^{\mathrm{r}}))$

Games $\boxed{\mathrm{G}_1}$, $\boxed{\mathrm{G}_2}$

$(f_0, St_{\mathrm{a}}) \leftarrow\!\!\$\ \mathcal{A}(\mathtt{s})$; $K^{\mathrm{r}} \leftarrow\!\!\$\ \mathsf{Aye}_{\mathrm{r}}.\mathsf{KS}$
$K^{\mathrm{f}} \leftarrow\!\!\$\ \mathsf{F}.\mathsf{KS}$; $\mathbf{A} \leftarrow f_0(K^{\mathrm{r}})$
For $\ell \in \mathbf{A}.\mathsf{Lbls}$ where $\ell = 0\|\ell'$ do
  $\boxed{K \leftarrow \mathsf{F}.\mathsf{Ev}(K^{\mathrm{f}}, \ell)}$; $\boxed{K \leftarrow\!\!\$\ \mathsf{SE}.\mathsf{KS}}$
  $\mathbf{A}[\ell] \leftarrow\!\!\$\ \mathsf{SE}.\mathsf{Enc}(K, \mathbf{A}[\ell])$
$(lk, St) \leftarrow\!\!\$\ \mathcal{L}(\mathtt{s}, \mathbf{A})$; $(ED, St') \leftarrow\!\!\$\ \mathcal{S}(\mathtt{s}, lk)$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{Tok}}(\mathtt{q}, ED, St_{\mathrm{a}})$; Return $b' = 1$

Oracle $\mathrm{Tok}(f)$

$(lk, St) \leftarrow\!\!\$\ \mathcal{L}(\mathtt{q}, f(K^{\mathrm{r}}), St)$
$(tk, St') \leftarrow\!\!\$\ \mathcal{S}(\mathtt{s}, lk, St')$; Return tk

Game G_3^m

$(f_0, St_{\mathrm{a}}) \leftarrow\!\!\$\ \mathcal{A}(\mathtt{s})$
$K^{\mathrm{r}} \leftarrow\!\!\$\ \mathsf{Aye}_{\mathrm{r}}.\mathsf{KS}$; $\mathbf{A} \leftarrow f_0(K^{\mathrm{r}})$
For $\ell \in \mathbf{A}.\mathsf{Lbls}$ where $\ell = 0\|\ell'$ do
  $K \leftarrow\!\!\$\ \mathsf{SE}.\mathsf{KS}$
  $\mathbf{A}[\ell] \leftarrow\!\!\$\ \mathsf{SE}.\mathsf{Enc}(K, \mathbf{A}[\ell])$
$(lk, St) \leftarrow\!\!\$\ \mathcal{L}(\mathtt{s}, \mathbf{A})$
$(ED, St') \leftarrow\!\!\$\ \mathcal{S}(\mathtt{s}, lk)$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{Tok}}(\mathtt{q}, ED, St_{\mathrm{a}})$; Return $b' = 1$

Oracle $\mathrm{Tok}(f)$

$n \leftarrow n + 1$; $q_n \leftarrow f(K^{\mathrm{r}})$; $x \leftarrow \min_{q_i = q_n} i$
$\big((X, lk), St\big) \leftarrow\!\!\$\ \mathcal{L}(\mathtt{q}, q_n, St)$
If $q_n = 0\|\ell$ then
  If $\mathbf{V}[x] = \perp$ then
    If $m < n$ then $\mathbf{V}[x] \leftarrow X$
      else $\mathbf{V}[x] \leftarrow\!\!\$\ \{0,1\}^{|X|}$
$(tk, St') \leftarrow\!\!\$\ \mathcal{S}(\mathtt{s}, (\mathbf{V}[x], lk), St')$
Return tk

Figure 3.8. Games used in proof of Theorem 20.

one can construct $\mathcal{A}_{\mathrm{f}}, \mathcal{A}_{\mathrm{se}}, \mathcal{A}_{\mathrm{st}}, \mathcal{S}_{\mathrm{f}}$ *such that:*

$$\mathbf{Adv}^{\mathcal{F}\text{-ss}}_{\mathsf{Aye}_{\mathrm{f}},\mathcal{L}_{\mathrm{f}},\mathcal{S}_{\mathrm{f}}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathcal{F}\text{-ss}}_{\mathsf{Aye}_{\mathrm{r}},\mathcal{L},\mathcal{S}}(\mathcal{A}_{\mathrm{ss}}) + \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{A}_{\mathrm{f}}) + M \cdot \mathbf{Adv}^{\mathrm{ind\$}}_{\mathsf{SE}}(\mathcal{A}_{\mathrm{se}}),$$

where M is the maximum number of response-hiding TOK *queries made by* $\mathcal{A}$*.*

Proof. We define the adversaries $\mathcal{A}_{\mathrm{ss}}, \mathcal{A}_{\mathrm{se}}, \mathcal{A}_{\mathrm{f}}$ and simulator $\mathcal{S}_{\mathrm{f}}$ in Fig. 3.7. Note that the simulator selects the $\mathbf{V}[x]$ values to be random strings of the appropriate length.

$\mathcal{A}_{\mathrm{ss}}$ modifies the function f_0 returned by $\mathcal{A}$ to also encrypt the response-hiding entries with SE. Note that the random coins for $\mathsf{SE.Enc}$ is selected by $\mathcal{A}_{\mathrm{ss}}$ and "hard-coded" into f_0^* since we expect functions to be deterministic.

$\mathcal{A}_{\mathrm{f}}$ simulates the entirety of the "ideal" game for $\mathcal{A}$ except that the generation of SE's keys using F is replaced with calls to its FN oracle.

$\mathcal{A}_{\mathrm{se}}$ performs the same simulation, but it generates the query response given to $\mathcal{S}(\mathsf{q},\cdot)$ in a few ways. For one randomly selected query, the ENC oracle is used. Encryptions (using SE) of $\mathbf{A}$'s values are used in the queries before this while random strings are used for those after. During this, we ensure that the query equality pattern is respected.

Now consider the hybrids in Fig. 3.8. Game G_0 captures what happens in the "real world" in both $\mathrm{G}^{\mathcal{F}\text{-ss}}_{\mathsf{Aye}_{\mathrm{f}},\mathcal{L}_{\mathrm{f}},\mathcal{S}_{\mathrm{f}}}(\mathcal{A})$ and $\mathrm{G}^{\mathcal{F}\text{-ss}}_{\mathsf{Aye}_{\mathrm{r}},\mathcal{L},\mathcal{S}}(\mathcal{A}_{\mathrm{ss}})$. G_1 uses the leakage algorithm and simulator for $\mathsf{Aye}_{\mathrm{r}}$ in place of $\mathsf{Aye}_{\mathrm{r}}$'s algorithms. G_2 differs from G_1 in how it selects the keys for SE, with the latter computing them with $\mathsf{F.Ev}$ and the former selecting them at random. G_3^m is defined for $m = 0, 1, \ldots, M$. For TOK queries before the m^{th} one, the query output is taken from $\mathcal{L}$, meaning it is the relevant value symmetrically encrypted under a random key. From the m^{th} one onward they are random strings. Notice that G_3^M is equivalent to G_2 since all the $\mathbf{V}[x]$ are generated using $\mathcal{L}$, and that G_3^0 captures what happens in the "ideal world" in $\mathrm{G}^{\mathcal{F}\text{-ss}}_{\mathsf{Aye}_{\mathrm{f}},\mathcal{L}_{\mathrm{f}},\mathcal{S}_{\mathrm{f}}}(\mathcal{A})$ where all the query outputs are random strings. From here, we can derive the bound by combining the following

equations (here, b is the challenge bit of the respective game):

$$\begin{aligned}
&\mathbf{Adv}^{\mathcal{F}\text{-ss}}_{\mathsf{Aye}_\mathrm{f},\mathcal{L}_\mathrm{f},\mathcal{S}_\mathrm{f}}(\mathcal{A}) = \Pr[\mathrm{G}^{\mathcal{F}\text{-ss}}_{\mathsf{Aye}_\mathrm{f},\mathcal{L}_\mathrm{f},\mathcal{S}_\mathrm{f}}(\mathcal{A}) = 1|b=1] - \Pr[\mathrm{G}^{\mathcal{F}\text{-ss}}_{\mathsf{Aye}_\mathrm{f},\mathcal{L}_\mathrm{f},\mathcal{S}_\mathrm{f}}(\mathcal{A}) = 1|b=0] \\
&\quad = \Pr[\mathrm{G}_0] - \Pr[\mathrm{G}^0_3] \\
&\Pr[\mathrm{G}_0] - \Pr[\mathrm{G}_1] = \Pr[\mathrm{G}^{\mathcal{F}\text{-ss}}_{\mathsf{Aye}_\mathrm{r},\mathcal{L},\mathcal{S}}(\mathcal{A}_\mathrm{ss}) = 1|b=1] - \Pr[\mathrm{G}^{\mathcal{F}\text{-ss}}_{\mathsf{Aye}_\mathrm{r},\mathcal{L},\mathcal{S}}(\mathcal{A}_\mathrm{ss}) = 1|b=0] \\
&\quad = \mathbf{Adv}^{\mathcal{F}\text{-ss}}_{\mathsf{Aye}_\mathrm{r},\mathcal{L},\mathcal{S}}(\mathcal{A}_\mathrm{ss}) \\
&\Pr[\mathrm{G}_1] - \Pr[\mathrm{G}_2] = \Pr[\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{A}_\mathrm{f}) = 1|b=1] - \Pr[\mathrm{G}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{A}_\mathrm{f}) = 1|b=0] \\
&\quad = \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{A}_\mathrm{f}) \\
&\Pr[\mathrm{G}_2] - \Pr[\mathrm{G}^0_3] = \Pr[\mathrm{G}^M_3] - \Pr[\mathrm{G}^0_3] = \sum_{i\in[M]} (\Pr[\mathrm{G}^i_3] - \Pr[\mathrm{G}^{i-1}_3]) \\
&\quad = \sum_{i\in[M]} \big(\Pr[\mathrm{G}^{\mathrm{ind\$}}_{\mathsf{SE}}(\mathcal{A}_\mathrm{se}) = 1|b=1, m=i] - \Pr[\mathrm{G}^{\mathrm{ind\$}}_{\mathsf{SE}}(\mathcal{A}_\mathrm{se}) = 1|b=0, m=i]\big) \\
&\quad = M \cdot \mathbf{Adv}^{\mathrm{ind\$}}_{\mathsf{SE}}(\mathcal{A}_\mathrm{se})
\end{aligned}$$

□

RfT preserves content obliviousness.

Let $\mathbf{A}_1, \mathbf{A}_2$ be homomorphic arrays, $\mathcal{L}_\mathrm{r}, \mathcal{L}_\mathrm{f}$ be the leakage algorithms of Aye_r and $\mathbf{RfT}[\mathsf{Aye}_\mathrm{r}, \mathsf{SE}, \mathsf{F}]$ respectively. Then, define:

$$(lk^i_0, St_i) \leftarrow\!\!\$\, \mathcal{L}_\mathrm{f}\big(\mathtt{s}, \mathbf{A}_i\big)\ ;\ (lk^i_1, St_i) \leftarrow\!\!\$\, \mathcal{L}_\mathrm{f}\big(\mathtt{q}, \ell_1, St_i\big)\ ;\ \ldots\ ;\ (lk^i_n, St_i) \leftarrow\!\!\$\, \mathcal{L}_\mathrm{f}\big(\mathtt{q}, \ell_n, St_i\big).$$

Notice that the $\mathbf{F}$ constructed in the $\mathcal{L}_\mathrm{f}\big(\mathtt{s}, \mathbf{A}_1\big), \mathcal{L}_\mathrm{f}\big(\mathtt{s}, \mathbf{A}_2\big)$ are homomorphic. This means that $lk^1_0 = lk^2_0$ since $\mathcal{L}_\mathrm{r}$ is content oblivious.

Suppose that $\ell_j = 1\|\ell$ and let $lk^i_j = (X_i, b\|x_i)$. Then, $x_1 = x_2$ because $\mathcal{L}_\mathrm{r}$ is content oblivious. Now suppose that $\ell_j = 0\|\ell$ and let $lk^i_j = (n_i, b\|x_i)$ for $i = 1, 2$. Since n_1, n_2 is derived from the query equality pattern (and has nothing to do with the input arrays) we have $n_1 = n_2$. At the same time, because $|\mathbf{A}_1[\ell_j]| = |\mathbf{A}_2[\ell_j]|$ then $\mathsf{SE.cl}(\mathbf{A}_1[\ell_j]) = \mathsf{SE.cl}(\mathbf{A}_2[\ell_j])$ which means

Adversary $\mathcal{A}_{\mathrm{f}}(\mathtt{s})$	**Function** $f_{\mathrm{tok}}^{\mathrm{TK}}$
$((\mathbf{P},\mathbf{I}),St_{\mathrm{a}}) \leftarrow\!\!\$\ \mathcal{A}(\mathtt{s})$	For $\ell \in \mathbf{P}.\mathsf{Lbls}$ do $\mathbf{F}[0\|\ell] \leftarrow \mathbf{P}[\ell]$
Define f_0: $f_0(K^{\mathrm{f}}) = f_{\mathrm{tok}}^{\mathsf{Aye_f}.\mathsf{Tok}(K^{\mathrm{f}},\cdot)}$	For all $\ell \in \mathbf{I}.\mathsf{Lbls}$ and $\ell' \in \mathbf{I}[\ell]$ do
Return (f_0, St_{a})	$\mathbf{F}[1\|\ell] \leftarrow \mathbf{F}[1\|\ell]\|\mathrm{TK}(0\|\ell')$
Adversary $\mathcal{A}_{\mathrm{f}}^{\mathrm{TOK}}(\mathtt{q},ED,St_{\mathrm{a}})$	Return $\mathbf{F}$
Return $\mathcal{A}^{\mathrm{TOK}^*}(\mathtt{q},ED,St_{\mathrm{a}})$	**Oracle** $\mathrm{TOK}^*(\ell)$
	Define f : $f(K^{\mathrm{f}}) = 1\|\ell$
	Return $\mathrm{TOK}(f)$

Figure 3.9. Adversary used in proof of Theorem 21.

that $|\mathbf{F}[\ell_j]|$ is the same no matter which $\mathbf{A}_i$ was used and $x_1 = x_2$. Therefore, in either case, the lk_j^i satisfy the condition for $\mathcal{L}_{\mathrm{f}}$ to be content oblivious.

Mon's security.

Similar to our composite solution, Mon achieves security under leakage algorithm $\mathcal{L}_{\mathrm{m}}$ if its primitive has content oblivious leakage. We first need to define what content obliviousness means for RF schemes. Intuitively, we require the same RR condition on queries $\ell = 1\|\ell'$ and on queries $\ell = 0\|\ell'$ the entirety of the leakage must match. More specifically, suppose RF AYE scheme $\mathsf{Aye_f}$ is content oblivious. Then, given any pair of homomorphic arrays $\mathbf{A}_1, \mathbf{A}_2$, labels $\ell_1, \ldots, \ell_n \in \mathbf{A}_1.\mathsf{Lbls}$, for $i = 1,2$ let

$$(lk_0^i, St_i) \leftarrow\!\!\$\ \mathcal{L}(\mathtt{s},\mathbf{A}_i)\ ;\ (lk_1^i, St_i) \leftarrow\!\!\$\ \mathcal{L}(\mathtt{q},\ell_1,St_i)\ ;\ \ldots\ ;\ (lk_n^i, St_i) \leftarrow\!\!\$\ \mathcal{L}(\mathtt{q},\ell_n,St_i).$$

Then, we require that:

$$\begin{aligned} \text{When } \ell_j = 0\|\ell'_j, &\quad lk_j^1 = lk_j^2, \\ \text{And when } \ell_j = 1\|\ell'_j, &\quad lk_j^1 = (\mathbf{A}_1[\ell'], lk_j) \text{ and } lk_j^2 = (\mathbf{A}_2[\ell'], lk_j) \end{aligned}$$

Then, if $\mathsf{Aye_f}$ is a $\mathcal{F}_{\mathrm{tok}}$-secure RF AYE with content oblivious leakage then $\mathsf{Mon} = \mathbf{MonT}[\mathsf{Aye_f}]$ is a $\mathcal{F}_{\mathrm{kInd}}$-secure StE (i.e. secure in the standard model) for DLdt with respect to

$\mathcal{L}_{\mathrm{m}}, \mathcal{S}_{\mathrm{m}}$ (as depicted in Fig. 3.5). We prove this below.

Theorem 21 *Let* $\mathsf{Aye}_{\mathrm{f}}$ *be a RF AYE scheme and* $\mathsf{Mon} = \mathbf{MonT}[\mathsf{Aye}_{\mathrm{f}}]$. *Then, given adversary* $\mathcal{A}$*, query-oblivious leakage algorithm* $\mathcal{L}_{\mathrm{f}}$ *and simulator* $\mathcal{S}_{\mathrm{f}}$*, one can construct* $\mathcal{A}_{\mathrm{f}}$ *such that*

$$\mathbf{Adv}^{\mathrm{ss}}_{\mathsf{Mon},\mathcal{L}_{\mathrm{m}},\mathcal{S}_{\mathrm{m}}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathcal{F}_{\text{tok}}\text{-ss}}_{\mathsf{Aye}_{\mathrm{f}},\mathcal{L}_{\mathrm{f}},\mathcal{S}_{\mathrm{f}}}(\mathcal{A}_{\mathrm{f}}).$$

where $\mathcal{L}_{\mathrm{m}}, \mathcal{S}_{\mathrm{m}}$ *are as defined in Fig. 3.5.*

Proof. The adversary $\mathcal{A}_{\mathrm{f}}$ is given in Fig. 3.9. It runs $\mathcal{A}$ converting its setup and query outputs into functions to play the $\mathrm{G}^{\mathcal{F}_{\text{tok}}\text{-ss}}_{\mathsf{Aye}_{\mathrm{f}},\mathcal{L}_{\mathrm{f}},\mathcal{S}_{\mathrm{f}}}$ game. The f_0^* it returns during the setup phase converts elements from $\mathsf{DLdt.Dom}$ into $\mathbf{F}$ in Mon, using the key only to construct tokens for the RR values, while the f returned during the query are constant functions. Note that this function tuple is in $\mathcal{F}_{\text{tok}}$, as desired. By the definition of $\mathcal{A}_{\mathrm{f}}$ and Mon, we have that

$$\Pr[\mathrm{G}^{\mathcal{F}_{\text{tok}}\text{-ss}}_{\mathsf{Aye}_{\mathrm{f}},\mathcal{L}_{\mathrm{f}},\mathcal{S}_{\mathrm{f}}}(\mathcal{A}_{\mathrm{f}}) = 1 | b = 1] = \Pr[\mathrm{G}^{\mathrm{ss}}_{\mathsf{Mon},\mathcal{L}_{\mathrm{m}},\mathcal{S}_{\mathrm{m}}}(\mathcal{A}) = 1 | b = 1]$$

where b is the challenge bit in the respective games.

As with Com, we use the content oblivious leakage assumption to ensure that the tokens returned by $\mathcal{S}_{\mathrm{m}}$ (which are RR queries to the $\mathbf{F}$) will return a set of tokens consistent with $\mathcal{S}_{\mathrm{f}}$'s output. This ensures that

$$\Pr[\mathrm{G}^{\mathcal{F}_{\text{tok}}\text{-ss}}_{\mathsf{Aye}_{\mathrm{f}},\mathcal{L}_{\mathrm{f}},\mathcal{S}_{\mathrm{f}}}(\mathcal{A}_{\mathrm{f}}) = 1 | b = 0] = \Pr[\mathrm{G}^{\mathrm{ss}}_{\mathsf{Mon},\mathcal{L}_{\mathrm{m}},\mathcal{S}_{\mathrm{m}}}(\mathcal{A}) = 1 | b = 0]$$

combining this with the above result we get the desired bound. □

Discussion.

We now elucidate the advantages of applications adopting our monolithic approach over the composite one.

First off, Mon leaks strictly less than Com using standard primitives. In particular, using standard schemes such as those in Fig. 3.2, Com will leak the number of blocks in each encrypted array while Mon will leak their sum. All other forms of leakage (i.e. query & access patterns) are comparable. In this way, Mon's leakage can be derived from Com (but not vice versa) indicating that it leaks strictly less. The reduction in leakage that comes with merging two data structures in StE for DLdt may seem insignificant, but if our technique is used in more complex systems the savings may be more substantial.

For example, if OPX adopts the monolithic approach it would merge $c^2 + c + 3$ data structures into a single array where c is the number of columns in a SQL database [92]. The individual setup leakage for each of these arrays can leak important information to an adversary. As an illustrative example, if standard AYE schemes are used, OPX will leak the distribution of join [5] sizes in the database because one array is created for each pair of columns $(\mathbf{c}, \mathbf{c}')$ to index their join. To see how this leakage could be abused, consider a database made up of two tables with one joinable pair of columns. If these columns contain boolean values (i.e. have a small domain), and we know from some meta information that the true and false values are distributed evenly, we can expect that the indexed join has a number of blocks proportional to $n_1 \cdot n_2/4$, where n_1 and n_2 are the number of rows in the two tables respectively. If instead the columns contain street addressses (i.e. large domain with sparse distribution), we would expect the indexed join's volume to instead be less than $\min(n_1, n_2)$. The adversary can therefore differentiate between the two databases using just the setup leakage, thereby learning something about what data the client is storing.

Aside from minimizing leakage, the monolithic setup (i.e. a single $\mathcal{F}_{\text{tok}}$-secure encrypted array) has more robust applications including a general multi-leveled indexed array which can store tokens referring back to itself in a response revealing way, until finally returning one or more response hidden entries to the client. The $\mathcal{F}_{\text{tok}}$-secure encrypted array already can provide security for this structure if one is willing to leak the access pattern or fix a constant access

[5] A join on columns $\mathbf{c}, \mathbf{c}'$ returns pairs of rows from their respective table where the column values in $\mathbf{c}, \mathbf{c}'$ match.

pattern for every query. In SPX, joins are handled in a way similar to a "three-level" indexed data structure [89]. It uses an encrypted dictionary to store encrypted multimaps, which store tokens to separate row multimaps. To handle this case, one could create one monolithic $\mathcal{F}_{\text{tok}}$-secure encrypted array that combines the dictionary and multimaps and into entries to the array. This would also simplify the construction to only require two queries into the array (since all top level multimaps would be combined removing the need for a dictionary query). Also, under the $\mathcal{F}_{\text{tok}}$-security notion, our scheme can already handle more complicated self-referencing than layered referencing with less leakage than a the composite solution.

3.6 KD-Secure StE for Broader Function Classes

In Section 3.5 we studied key-dependent (KD) StE with respect to the token generating function class $\mathcal{F}_{\text{tok}}$ in detail. We believe our KD StE notion is of independent interest and extend it in this section via broader function classes; namely the set of all functions $\mathcal{F}_{\text{all}}$ and the more restrictive class of "fixed-format" functions $\mathcal{F}_{\text{ff}}$.

While our discussion focuses on KD-security for AYE schemes, we also encounter subtle issues when bringing together notions of KD and semantic security which apply to StE schemes in general. More specifically, we address KD-security for RR and RH AYE schemes separately, bringing up a new game in each modeling non-standard behavior. We show that with the exception of these boundary cases, out of the four possible types of KD-secure AYE, only $\mathcal{F}_{\text{ff}}$-secure RH AYE can be efficiently achieved. We then give (generic and dedicated) transforms to construct this from KDM-secure symmetric encryption (SE). We conclude by extending all of these results to RF AYE.

Fixed-format functions.

Recall that in Section 3.5 we defined $\mathcal{F}_{\text{all}}$ as the set of all $(f_0, \ldots, f_n)$ which the adversary in the KD-security game can provide. Applied to AYE, this is the universal function class allowing the adversary to provide any (key-dependent) array and sequence of labels that it

Alg $\mathcal{A}_{\mathrm{r}}(\mathsf{s})$	**Game** $\mathrm{G}^{\mathrm{tok}}_{\mathsf{StE},\mathcal{L},\mathcal{S}}(\mathcal{A})$
Define $f_0 : f_0(K) = \mathbf{A}$ where	$(DS, St_{\mathrm{a}}) \leftarrow\!\!\$\ \mathcal{A}(\mathsf{s})$; Require $DS \in \mathsf{DT.Dom}$
$\quad \mathbf{A}[0] \leftarrow \varepsilon$; $\mathbf{A}[1] \leftarrow \langle K \rangle_{\mathsf{bLen}}$	$(lk, St) \leftarrow \mathcal{L}(\mathsf{s}, DS)$; $(ED, St') \leftarrow \mathcal{S}(\mathsf{s}, lk)$
Return (f_0, St_{a})	$tk \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{Tok}}(\mathsf{q}, ED, St_{\mathrm{a}})$; Require $tk \notin T$
Alg $\mathcal{A}_{\mathrm{r}}^{\mathrm{Tok}}(\mathsf{q}, ED, St_{\mathrm{a}})$	Return $\mathsf{StE.Eval}(tk, ED) \neq \bot$
$tk_1 \leftarrow\!\!\$\ \mathrm{Tok}(1)$	**Oracle** $\mathrm{Tok}(q)$
$K \leftarrow \mathsf{Aye}_{\mathrm{r}}.\mathsf{Eval}(tk_1, ED)$	Require $q \in \mathsf{DT.QS}$
$tk_0 \leftarrow\!\!\$\ \mathsf{Aye}_{\mathrm{r}}.\mathsf{Tok}(K, 0)$	$(lk, St) \leftarrow\!\!\$\ \mathcal{L}(\mathsf{q}, q, St)$; $(tk, St') \leftarrow\!\!\$\ \mathcal{S}(\mathsf{q}, lk, St')$
Return tk_0	$T \xleftarrow{\cup} tk$; Return tk

Figure 3.10. Key-storing adversary used in Theorem 22 (left) and token forgery game (right) where StE is an StE scheme for DT with leakage algorithm $\mathcal{L}$ and simulator $\mathcal{S}$.

chooses. We define the set of *fixed-format* functions $\mathcal{F}_{\mathrm{ff}}$ to be a subclass of this.

Intuitively, fixed-format functions provide arrays that are homomorphic (as defined in Section 3.3) and queries that are key-independent. Specifically, if $f = (f_0, f_1, \ldots, f_n) \in \mathcal{F}_{\mathrm{ff}}$, then $f \in \mathcal{F}_{\mathrm{all}}$ and for all K_1, K_2 in Aye.KS, we have that $f_0(K_1), f_0(K_2)$ are homomorphic and $f_i(K_1) = f_i(K_2)$ for all $i \in [n]$.

An equivalent definition involves capturing f_0 in terms of fixed-length per-label functions. In other words, if $(f_0, f_1, \ldots, f_n) \in \mathcal{F}_{\mathrm{ff}}$ then there exists a fixed (key-independent) label set $L \subseteq \{0,1\}^*$ where $L = f_0(K).\mathsf{Lbls}$ and there exists functions $\{g_\ell\}_{\ell \in L}$ such that:

$$f_0(K) = \mathbf{A} \ \text{ where } \ \mathbf{A}[\ell] = \begin{cases} g_\ell(K) & \text{if } \ell \in L \\ \bot & \text{otherwise.} \end{cases}$$

Additionally, for all $K_1, K_2 \in \mathsf{Aye.KS}$, we require that $|g_\ell(K_1)| = |g_\ell(K_2)|$ for all $\ell \in L$ and $f_i(K_1) = f_i(K_2)$ for all $i \in [n]$. This "broken down" version of the f_0 is particularly useful in our reductions.

3.6.1 KD-security of Response-Revealing AYE

Key-storing adversary.

Intuitively, $\mathcal{F}_{\text{ff}}$-secure RR AYE should be impossible because the adversary can use f_0 to store the encryption key as a value in the array. During the query phase, the adversary requests a token for this array value and can retrieve this key from the encrypted data structure because the scheme is response-revealing. This should void any meaningful notion of KD-security for AYE. This adversary is formalized as $\mathcal{A}_{\text{r}}$ in on the left side of Fig. 3.10. Note that the tuple of functions provided by $\mathcal{A}_{\text{r}}$ is in $\mathcal{F}_{\text{ff}}$.

However, pinning down Aye_{r}'s advantage is not straightforward and highlights a subtle issue of boundary cases in notions of semantic security which applies to StE in general. As an illustrative example, consider the "trivial" scheme, leakage algorithm and simulator for DT that we define as follows:

$$\mathsf{StE}_{\text{t}}.\mathsf{Enc}(K,DS) = DS,\ \mathsf{StE}_{\text{t}}.\mathsf{Tok}(K,q) = q,\ \mathsf{StE}_{\text{t}}.\mathsf{Eval}(q,DS) = \mathsf{DT}.\mathsf{Spec}(DS,q),$$
$$\mathcal{L}_{\text{t}}\big(\mathtt{s},DS\big) = \mathcal{S}_{\text{t}}\big(\mathtt{s},DS\big) = (DS,\varepsilon),\ \mathcal{L}_{\text{t}}\big(\mathtt{q},q,\varepsilon\big) = \mathcal{S}_{\text{t}}\big(\mathtt{s},1,\varepsilon\big) = (q,\varepsilon).$$

Then, notice that $\mathcal{F}_{\text{ff}}$-security is possible for StE_{t} with respect to $\mathcal{L}_{\text{t}},\mathcal{S}_{\text{t}}$ because the leakage algorithm reveals everything to the simulator. In order to demonstrate the devastating effect of the key-storing adversary we need a formal notion which allows us to rule out excessively leaky algorithms like these. We do this with the token forgery game below.

Token forgery game.

The token forgery game $\text{G}^{\text{tok}}_{\mathsf{StE},\mathcal{L},\mathcal{S}}$ is depicted in Fig. 3.10. The game is similar to $\text{G}^{\text{ss}}_{\mathsf{StE},\mathcal{L},\mathcal{S}}$ when the challenge bit is 0 meaning that ED and the tk returned by the token oracle are simulated. The adversary's goal is to forge a usable token meaning it wins by returning a token (different from any output of the token oracle) which does not return $\perp$ when evaluated with $\mathsf{StE}.\mathsf{Eval}$ and

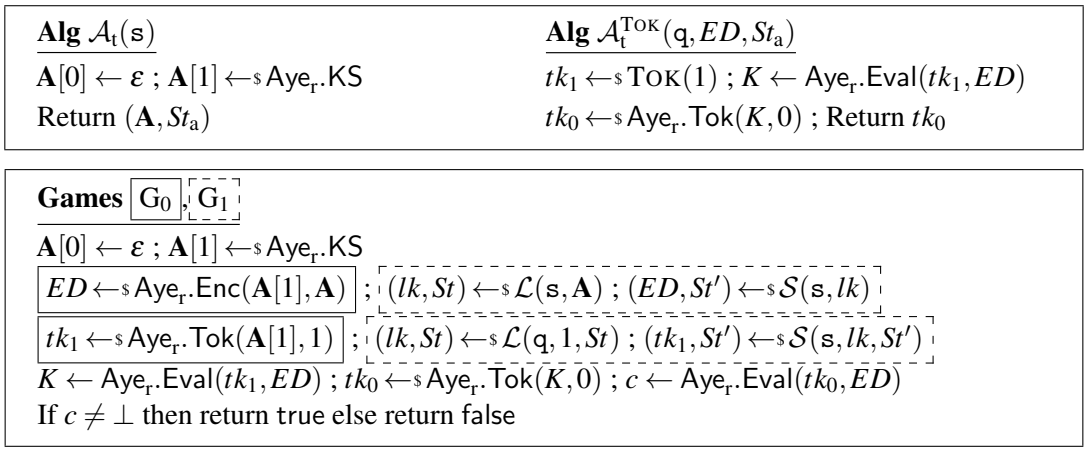

Figure 3.11. Adversaries and games used in proof of Theorem 22. $\mathsf{Aye}_{\mathrm{r}}$ is a RR AYE scheme, $\mathcal{L}$ is a leakage algorithm and $\mathcal{S}$ is a simulator.

ED. We define the token forgery advantage of $\mathcal{A}$ as $\mathbf{Adv}^{\mathrm{tok}}_{\mathsf{StE},\mathcal{L},\mathcal{S}}(\mathcal{A}) = \Pr[\mathrm{G}^{\mathrm{tok}}_{\mathsf{StE},\mathcal{L},\mathcal{S}}(\mathcal{A})]$. Notice that with the trivial scheme, tokens are easily forged since queries are passed to the server in the clear.

$\mathcal{F}_{\mathrm{ff}}$-secure RR AYE is (essentially) impossible.

With the above formalism, we show that any RH AYE scheme either permits a token-forging adversary with high advantage or is such that the key-storing adversary $\mathcal{A}_{\mathrm{r}}$ has a high advantage in the $\mathcal{F}_{\mathrm{ff}}$-security game. This is captured in the theorem below.

Theorem 22 *Let* $\mathsf{Aye}_{\mathrm{r}}$ *be a RR AYE scheme with leakage algorithm* $\mathcal{L}$ *and simulator* $\mathcal{S}$*. Let* $\mathcal{A}_{\mathrm{r}}$ *be as described in Fig. 3.10. Then, one can construct adversary* $\mathcal{A}_{\mathrm{t}}$ *such that:*

$$\mathbf{Adv}^{\mathcal{F}_{\mathrm{ff}}\text{-}\mathrm{ss}}_{\mathsf{Aye}_{\mathrm{r}},\mathcal{L},\mathcal{S}}(\mathcal{A}_{\mathrm{r}}) + \mathbf{Adv}^{\mathrm{tok}}_{\mathsf{Aye}_{\mathrm{r}},\mathcal{L},\mathcal{S}}(\mathcal{A}_{\mathrm{t}}) \geq 1$$

Proof. We give hybrid games $\mathrm{G}_0, \mathrm{G}_1$ in Fig. 3.11. They differ in whether $\mathsf{Aye}_{\mathrm{r}}$'s algorithms or $\mathcal{L}, \mathcal{S}$ are used to generate ED and token tk_1. Let b be the challenge bit in $\mathrm{G}^{\mathcal{F}_{\mathrm{ff}}\text{-}\mathrm{ss}}_{\mathsf{Aye}_{\mathrm{r}},\mathcal{L},\mathcal{S}}$. Then, by our

Game $\mathrm{G}^{\mathrm{ex}}_{\mathsf{StE}}(\mathcal{A})$	**Alg** $\mathcal{A}_{\mathrm{h}}(\mathtt{s})$
$K \leftarrow\!\!\$\ \mathsf{StE.KS}$; $f_0 \leftarrow\!\!\$\ \mathcal{A}(\mathtt{s})$	$(f_0, St_{\mathrm{a}}) \leftarrow\!\!\$\ \mathcal{A}_{\mathrm{e}}(\mathtt{s})$
$DS \leftarrow f_0(K)$	Select ℓ such that $\ell \notin f_0(K).\mathsf{Lbls}$ for all K
Require $DS \in \mathsf{DT.Dom}$	Define $f_0^* : f_0^*(K) = \mathbf{A}$ where
$ED \leftarrow\!\!\$\ \mathsf{StE.Enc}(K, DS)$	$\mathbf{A} \leftarrow f_0(K)$; $\mathbf{A}[\ell] \leftarrow \varepsilon$
$K' \leftarrow\!\!\$\ \mathcal{A}_{\mathrm{e}}^{\mathrm{TOK}}(\mathtt{q}, ED)$	Return $\left(f_0^*, (St_{\mathrm{a}}, \ell)\right)$
Return $K = K'$	**Alg** $\mathcal{A}_{\mathrm{h}}^{\mathrm{TOK}}(\mathtt{q}, ED, (St_{\mathrm{a}}, \ell))$
Oracle $\mathrm{TOK}(f)$	$K \leftarrow\!\!\$\ \mathcal{A}_{\mathrm{e}}^{\mathrm{TOK}}(\mathtt{q}, ED)$
$q \leftarrow f(K)$; Require $q \in \mathsf{DT.QS}$	$tk \leftarrow\!\!\$\ \mathsf{Aye_h.Tok}(K, \ell)$; $c \leftarrow \mathsf{Aye_h.Eval}(tk, ED)$
Return $\mathsf{StE.Tok}(K, q)$	If $c \neq \perp$ then return 1 else return 0

Figure 3.12. Key-exfiltration game (left) where StE is an StE scheme for DT and key-retrieving adversary $\mathcal{A}_{\mathrm{h}}$ (right) which runs key-exfiltration adversary $\mathcal{A}_{\mathrm{e}}$.

definition of $\mathcal{A}_{\mathrm{r}}$,

$$\begin{aligned}\mathbf{Adv}^{\mathcal{F}_{\text{ff-ss}}}_{\mathsf{Aye_r},\mathcal{L},\mathcal{S}}(\mathcal{A}_{\mathrm{r}}) &= \Pr[\mathrm{G}^{\mathcal{F}_{\text{ff-ss}}}_{\mathsf{Aye_r},\mathcal{L},\mathcal{S}}(\mathcal{A}_{\mathrm{r}}) = 1 | b = 1] - \Pr[\mathrm{G}^{\mathcal{F}_{\text{ff-ss}}}_{\mathsf{Aye_r},\mathcal{L},\mathcal{S}}(\mathcal{A}_{\mathrm{r}}) = 1 | b = 0] \\ &= \Pr[\mathrm{G}_0] - \Pr[\mathrm{G}_1].\end{aligned}$$

$\mathcal{A}_{\mathrm{t}}$ is also given in Fig. 3.11. In the setup phase, $\mathcal{A}_{\mathrm{t}}(\mathtt{s})$ constructs an array similar to the one in $\mathcal{A}_{\mathrm{r}}(\mathtt{s})$ Notice also that because $\mathcal{A}_{\mathrm{r}}$ is a response-revealing scheme, by the correctness condition $\Pr[\mathrm{G}_0] = 1$. Additionally, $\mathrm{G}^{\mathrm{tok}}_{\mathsf{Aye_r},\mathcal{L},\mathcal{S}}(\mathcal{A}_{\mathrm{t}})$ is equivalent to G_1 so $\Pr[\mathrm{G}_1] = \mathbf{Adv}^{\mathrm{tok}}_{\mathsf{Aye_r},\mathcal{L},\mathcal{S}}(\mathcal{A}_{\mathrm{t}})$. Combining this with the above we get the bound in the theorem. □

We note that practical RR AYE schemes would not allow token forgery (else it achieves no meaningful security). This means that $\mathcal{F}_{\text{ff}}$-secure RR AYE is impossible. Note that this implies $\mathcal{F}_{\text{all}}$-secure RR AYE is also (essentially) impossible.

3.6.2 KD-security of Response-Hiding AYE

Handling RH AYE is substantially different from RR AYE because the server never has the chance to see "unencrypted" data. This means that RH AYE bears many more parallels to BRS' notion of KDM-secure symmetric encryption (SE): just as KDM SE is only possible when

functions are fixed length, RR AYE can only efficiently achieve $\mathcal{F}_{\text{ff}}$-security and not $\mathcal{F}_{\text{all}}$-security. We discuss these positive and negative results in this subsection, along with the nuances therein.

Key-retrieval in RH AYE.

As an illustrative example, let Aye_{h} be a scheme which leaks whether a query returns $\perp$ in the following way:

$$\mathbf{A}[\ell] = \perp \quad \iff \quad \mathsf{Aye}_{\text{h}}.\mathsf{Eval}\big(\mathsf{Aye}_{\text{h}}.\mathsf{Tok}(K,\ell), \mathsf{Aye}_{\text{h}}.\mathsf{Enc}(K,\mathbf{A})\big) = \perp.$$

Then, an adversary playing the $\mathrm{G}^{\mathcal{F}_{\text{all}}\text{-ss}}_{\mathsf{Aye}_{\text{h}},\mathcal{L},\mathcal{S}}$ game can retrieve a secret key of length λ by picking distinct labels $\ell_1,\ldots,\ell_\lambda$ and value $v \neq \perp$ then defining $(f_0, f_1, \ldots, f_\lambda) \in \mathcal{F}_{\text{all}}$ such that f_0 sets $\mathbf{A}[\ell_i] \leftarrow v$ if the k^{th} bit of K is 1 and $\mathbf{A}[\ell_i] \leftarrow \perp$ otherwise. During the query phase, it requests tokens for $\ell_1,\ldots,\ell_\lambda$ and evaluates each with ED to retrieve, bit-by-bit, the key K. We note that RH AYE schemes with "standard" leakage (including $\mathsf{Aye}^{\pi}_{\text{h}}$ in Fig. 3.2) satisfy the above condition and are susceptible to this attack.

Once again, formalizing this intuition into an adversary and advantage is not straightforward for two reasons. First, notice that key retrieval does not help the adversary attacking trivial AYE primitives (i.e. the RH AYE variant of $\mathsf{StE}_{\text{t}}, \mathcal{L}_{\text{t}}, \mathcal{S}_{\text{t}}$ above, where $\mathsf{StE}_{\text{t}}.\mathsf{Dec}(K,c) = c$) because the schemes do not use the key at all. Therefore, more generally, we once again need to rule out excessively leaky Aye_{h}.

The second issue has to do with generalizing the key-retrieval mechanism in the attack. Note that even if Aye_{h} were to avoid leaking whether a query returns $\perp$ (e.g. by padding the array with dummy values), a similar attack could still be launched using other forms of leakage such as query equality pattern or response lengths. We want our key-retrieval adversary to capture the whole spectrum of such attacks and do so by defining a key-exfiltration game.

Key exfiltration game.

The key-exfiltration game $\mathrm{G}^{\text{ex}}_{\mathsf{StE}}$ is defined in Fig. 3.12. It is very similar to the $\mathcal{F}_{\text{all}}$-

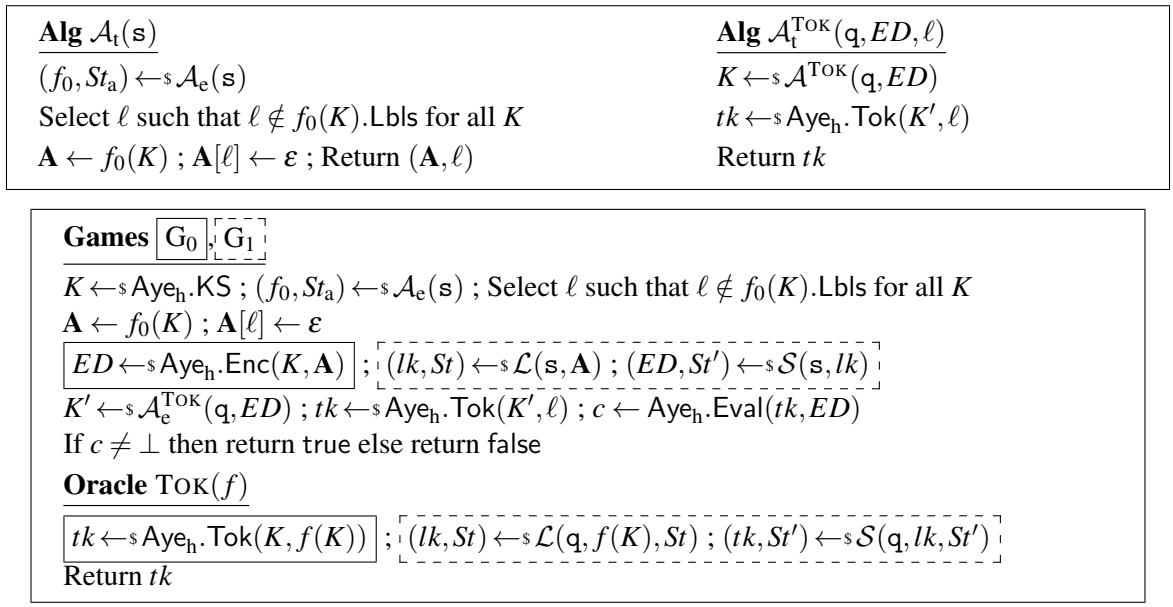

Figure 3.13. Adversaries and games used in in the proof of Theorem 23 where $\mathcal{A}_e$ is a key-exfiltration adversary for RH AYE scheme Aye_h, $\mathcal{L}$ is a leakage algorithm and $\mathcal{S}$ is a simulator.

security StE game when the challenge bit is 1 except that the adversary's goal is to retrieve the secret key. We define the key-exfiltration advantage of $\mathcal{A}$ as $\mathbf{Adv}^{\text{ex}}_{\mathsf{StE}}(\mathcal{A}) = \Pr[\text{G}^{\text{ex}}_{\mathsf{StE}}(\mathcal{A})]$. Note that due to the "Require" statements in the game, the adversary must provide a tuple of functions in $\mathcal{F}_{\text{all}}$.

Note that the adversary sketched above (which abuses queries returning $\perp$) can be captured as a key-retrieval adversary for Aye_h. Its advantage is 1 by Aye_h's correctness condition.

Key-retrieving adversary.

Given any key-exfiltration adversary $\mathcal{A}_e$ (which plays $\text{G}^{\text{ex}}_{\mathsf{StE}}$), we can construct a key-retrieval adversary $\mathcal{A}_h$ (which plays $\text{G}^{\mathcal{F}_{\text{all}}\text{-ss}}_{\mathsf{StE},\mathcal{L},\mathcal{S}}$) as shown in Fig. 3.12. Intuitively, $\mathcal{A}_h$ runs $\mathcal{A}_e$ and then "tests" the exfiltrated key by generating a token and evaluating it. We evaluate the advantage of $\mathcal{A}_h$ in the following result:

Theorem 23 *Let* Aye_h *be a RH AYE scheme with leakage algorithm* $\mathcal{L}$ *and simulator* $\mathcal{S}$*. Let* $\mathcal{A}_h$ *be as described in Fig. 3.13 for some key-exfiltration adversary* $\mathcal{A}_e$*. Then, one can construct* $\mathcal{A}_t$

such that:

$$\mathbf{Adv}^{\mathcal{F}_{\text{all-ss}}}_{\mathsf{Aye}_\text{h},\mathcal{L},\mathcal{S}}(\mathcal{A}_\text{h}) + \mathbf{Adv}^{\text{tok}}_{\mathsf{Aye}_\text{h},\mathcal{L},\mathcal{S}}(\mathcal{A}_\text{t}) \geq \mathbf{Adv}^{\text{ex}}_{\mathsf{Aye}_\text{h}}(\mathcal{A}_\text{e})$$

Proof. We give hybrid games G_0, G_1 in Fig. 3.13. In the former, encryption and token generation are done with Aye_h's algorithms while in the latter it is done with $\mathcal{L}, \mathcal{S}$ (with the exception of the last tk). Both run $\mathcal{A}_\text{e}$ and win when $\mathsf{Aye}_\text{h}.\mathsf{Eval}(tk, ED)$ does not return $\perp$. Then, let b be the challenge bit in $\text{G}^{\mathcal{F}_{\text{all-ss}}}_{\mathsf{Aye}_\text{h},\mathcal{L},\mathcal{S}}$ and notice that:

$$\begin{aligned}
\mathbf{Adv}^{\mathcal{F}_{\text{all-ss}}}_{\mathsf{Aye}_\text{h},\mathcal{L},\mathcal{S}}(\mathcal{A}_\text{h}) &= \Pr[\text{G}^{\mathcal{F}_{\text{all-ss}}}_{\mathsf{Aye}_\text{h},\mathcal{L},\mathcal{S}}(\mathcal{A}_\text{h}) = 1 | b = 1] - \Pr[\text{G}^{\mathcal{F}_{\text{all-ss}}}_{\mathsf{Aye}_\text{h},\mathcal{L},\mathcal{S}}(\mathcal{A}_\text{h}) = 1 | b = 0] \\
&= \Pr[\text{G}_0] - \Pr[\text{G}_1] \\
&= \Pr[\text{G}_0] - \mathbf{Adv}^{\text{tok}}_{\mathsf{Aye}_\text{h},\mathcal{L},\mathcal{S}}(\mathcal{A}_\text{t}).
\end{aligned}$$

The last line comes from noticing that G_1 also depicts what happens in $\text{G}^{\text{tok}}_{\mathsf{StE},\mathcal{L},\mathcal{S}}(\mathcal{A}_\text{t})$.

Now let K, K' be the random variables in the pseudocode of G_0. Notice that when $K = K'$, the adversary should always win because of Aye_h's correctness condition. Notice also that the probability that $K = K'$ in G_0 is exactly the key-exfiltration advantage of $\mathcal{A}_\text{e}$. Therefore,

$$\begin{aligned}
\Pr[\text{G}_0] &= \Pr[\text{G}_0 | K = K'] \Pr[K = K'] + \Pr[\text{G}_0 \wedge (K \neq K')] \\
&\geq \Pr[K = K'] = \mathbf{Adv}^{\text{ex}}_{\mathsf{Aye}_\text{h}}(\mathcal{A}_\text{e}).
\end{aligned}$$

Combining this with the above equation gives us the desired bound. □

$\mathcal{F}_{\text{all}}$-secure RH AYE is expensive.

The above theorem shows that if there exists an adversary with high exfiltration advantage, then Aye_h is insecure (for at least one of two reasons). This is particularly problematic because the leakage profiles of "standard" schemes elicit such exfiltration attacks. We sketch how two forms of leakage in such schemes allow for key exfiltration.

The first example of abusable leakage is query equality pattern. Any scheme which

reveals equivalent pairs of queries has an adversary with high key-exfiltration advantage. The adversary submits an array with at least two unique labels, call two of them ℓ_0 and ℓ_1, in its submitted array. Then, for each query f_i, the adversary submits a function which requests the token for ℓ_{k_i} where k_i is the ith bit of the key. Finally, the adversary queries ℓ_0. If the RH AYE scheme reveals query equality, then the adversary can determine which bits of the key are 0 by looking at the equality to the final query. In order to avoid this leakage, PPY show [6] that one must incur at least a logarithmic bandwidth overhead [113], at which point one could use ORAM to hide any access pattern from the adversary. This result shows that any constant overhead RH AYE schemes are vulnerable to high key-exfiltration advantage.

Our second example extends the illustrative one given earlier which abuses volume leakage (i.e. the number of blocks returned by a query). We can design an adversary, similar to the last one, which submits an array with at least two labels, ℓ_0 and ℓ_1, which map to values of different lengths (e.g. $|\mathbf{A}[\ell_0]| = \mathsf{bLen}$ and $|\mathbf{A}[\ell_1]| = 2 \cdot \mathsf{bLen}$). Again, for each query f_i, the adversary submits a function which requests the token for ℓ_{k_i} where k_i is the ith bit of the key. Finally, the adversary queries ℓ_0. Then, we can compare the lengths of each of these queries to the final query to determine the bits of the key. Alternatively, we could avoid key-dependent queries by choosing a label for each i and picking its length as long or short based on the value of k_i, then recovering this later. One can avoid this leakage using volume-hiding primitives, which may incur significant bandwidth overhead if the maximum value length in the array greatly exceeds that of most other values. Volume-hiding primitives in the literature also incur significant storage overhead [90, 114].

More generally, the necessary conditions under for key exfiltration not to occur is comparable KMO's notion of "zero-leakage" (ZL) primitives [91], which limits leakage to the security parameter and public information. Confirming this intuition is the fact that ZL schemes also incur the logarithmic bandwidth overhead of ORAM and worst-case storage of

[6] They prove a lower bound for the slightly weaker notion of decoupled query equality, which is enough to run the adversary described.

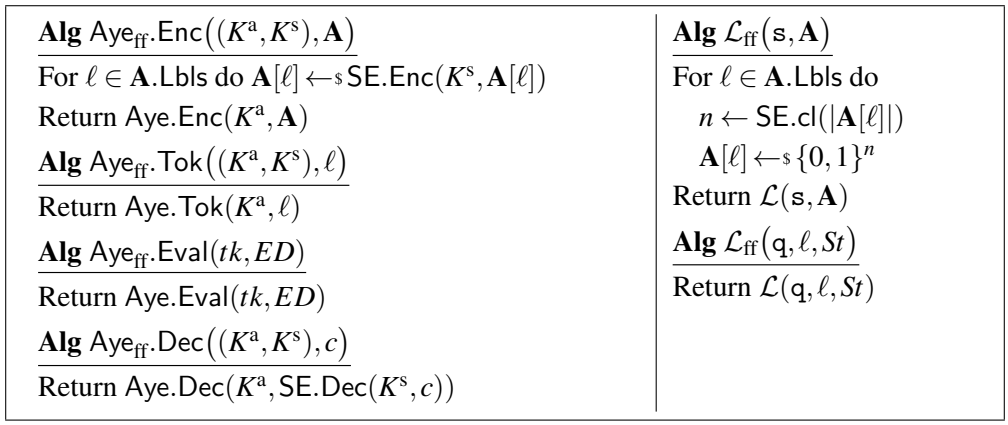

Figure 3.14. Algorithms (left) and leakage algorithm (right) for $\mathcal{F}_{\mathrm{ff}}$-secure RH AYE scheme $\mathsf{Aye}_{\mathrm{ff}} = \mathbf{FfT}[\mathsf{Aye},\mathsf{SE}]$ constructed using AYE scheme Aye (with leakage algorithm $\mathcal{L}$) and KDM-secure SE scheme SE.

volume-hiding primitives. But the notions are not exactly the same since an adversary could exfiltrate key bits using the ZL scheme's "public" information such as the maximum value length and total number of blocks in the array. At the same time, given a sufficiently long key (where 2^{λ} is greater than the number of distinguishable leakage profiles) a scheme may not be ZL but still won't permit key-exfiltration of the whole key.

In conclusion, $\mathcal{F}_{\mathrm{all}}$-secure RH AYE is impossible for "standard" schemes and requires significant bandwidth, storage and key-management overhead to achieve.

Generic RH transform for $\mathcal{F}_{\mathrm{ff}}$-secure AYE.

On the other hand, with RH AYE $\mathcal{F}_{\mathrm{ff}}$-security is achievable. We give a transform **FfT** which takes a AYE scheme Aye and SE scheme SE and returns an RH AYE scheme. The goal here is to "wrap" the $\mathbf{A}$ values with a layer of KDM SE, thereby elevating the standard security (i.e. $\mathcal{F}_{\mathrm{kInd}}$-security) of Aye to $\mathcal{F}_{\mathrm{ff}}$-security. The detailed algorithms for $\mathsf{Aye}_{\mathrm{ff}} = \mathbf{FfT}[\mathsf{Aye},\mathsf{SE}]$ are given in Fig. 3.14. Note that $\mathsf{Aye}_{\mathrm{ff}}.\mathsf{KS} = \mathsf{Aye}.\mathsf{KS} \times \mathsf{SE}.\mathsf{KS}$. We state and prove its security below.

Theorem 24 *Let $\mathcal{A}$ be an adversary, $\mathcal{L}$ be a leakage algorithm and $\mathcal{S}$ be a simulator. Let $\mathsf{Aye}_{\mathrm{ff}} = \mathbf{FfT}[\mathsf{Aye},\mathsf{SE}]$ be the RH AYE scheme and $\mathcal{L}_{\mathrm{ff}}$ be the leakage algorithm in Fig. 3.14. Then,*

Adversary $\mathcal{A}^{\text{Enc}}_{\text{se}}$

$K^{\text{a}} \leftarrow\!\!\$\ \mathsf{Aye.KS}$; $(f_0, St_{\text{a}}) \leftarrow\!\!\$\ \mathcal{A}(\mathtt{s})$
Define $L, \{g_\ell\}_{\ell \in L}$: as in Sect. 3.6 for f_0
For $\ell \in L$ do
  Define g' : $g'(K^{\text{s}}) = g_\ell(K^{\text{a}})$
  $\mathbf{A}[\ell] \leftarrow\!\!\$\ \text{Enc}(g')$
$ED \leftarrow\!\!\$\ \mathsf{Aye.Enc}(K^{\text{a}}, \mathbf{A})$
Return $\mathcal{A}^{\text{Tok}^*}(\mathtt{q}, ED, St_{\text{a}})$

Oracle $\text{Tok}^*(f)$

Return $\mathsf{Aye.Tok}\big(K^{\text{a}}, f(K^{\text{a}})\big)$

Adversary $\mathcal{A}_{\text{st}}(\mathtt{s})$

$(f_0, St_{\text{a}}) \leftarrow\!\!\$\ \mathcal{A}(\mathtt{s})$
Define $L, \{g_\ell\}_{\ell \in L}$: as in Sect. 3.6 for f_0
For $\ell \in L$ do $\mathbf{A}[\ell] \leftarrow\!\!\$\ \{0,1\}^{\mathsf{SE.cl}(|g_\ell(K^{\text{a}})|)}$
Return $(\mathbf{A}, St_{\text{a}})$

Adversary $\mathcal{A}^{\text{Tok}}_{\text{st}}(\mathtt{q}, ED, St_{\text{a}})$

Return $\mathcal{A}^{\text{Tok}^*}(\mathtt{q}, ED, St_{\text{a}})$

Oracle $\text{Tok}^*(f)$

Define x : $x = f(K^{\text{a}})$
Return $\text{Tok}(x)$

Games $\boxed{\text{G}_0}$, $\boxed{\text{G}_1}$

$K^{\text{a}} \leftarrow\!\!\$\ \mathsf{Aye.KS}$; $(f_0, St_{\text{a}}) \leftarrow\!\!\$\ \mathcal{A}(\mathtt{s})$
Define $L, \{g_\ell\}_{\ell \in L}$: as in Sect. 3.6 for f_0
For $\ell \in L$ do
  $\boxed{\mathbf{A}[\ell] \leftarrow\!\!\$\ \mathsf{SE.Enc}(g_\ell(K^{\text{a}}))}$
  $\boxed{\mathbf{A}[\ell] \leftarrow\!\!\$\ \{0,1\}^{\mathsf{SE.cl}(|g_\ell(K^{\text{a}})|)}}$
$ED \leftarrow\!\!\$\ \mathsf{Aye.Enc}(K^{\text{a}}, \mathbf{A})$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\text{Tok}}(\mathtt{q}, ED, St_{\text{a}})$; Return $b' = 1$

Oracle $\text{Tok}(f)$

Return $\mathsf{Aye.Tok}(f(K^{\text{a}}))$

Game G_2

$K^{\text{a}} \leftarrow\!\!\$\ \mathsf{Aye.KS}$; $(f_0, St_{\text{a}}) \leftarrow\!\!\$\ \mathcal{A}(\mathtt{s})$
Define $L, \{g_\ell\}_{\ell \in L}$: as in Sect. 3.6 for f_0
For $\ell \in L$ do $\mathbf{A}[\ell] \leftarrow\!\!\$\ \{0,1\}^{\mathsf{SE.cl}(|g_\ell(K^{\text{a}})|)}$
$(lk, St) \leftarrow\!\!\$\ \mathcal{L}(\mathtt{s}, \mathbf{A})$
$(ED, St') \leftarrow\!\!\$\ \mathcal{S}(\mathtt{s}, lk)$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\text{Tok}}(\mathtt{q}, ED, St_{\text{a}})$; Return $b' = 1$

Oracle $\text{Tok}(f)$

$(lk, St) \leftarrow\!\!\$\ \mathcal{L}(\mathtt{q}, f(K^{\text{a}}), St)$
$(tk, St') \leftarrow\!\!\$\ \mathcal{S}(\mathtt{q}, lk, St')$; Return tk

Figure 3.15. Adversaries and games used in proof of Theorem 24.

there exists $\mathcal{A}_{\text{st}}, \mathcal{A}_{\text{se}}$ *such that:*

$$\mathbf{Adv}^{\mathcal{F}_{\text{ff}}\text{-ss}}_{\mathsf{Aye}_{\text{ff}},\mathcal{L}_{\text{ff}},\mathcal{S}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{ss}}_{\mathsf{Aye},\mathcal{L},\mathcal{S}}(\mathcal{A}_{\text{st}}) + \mathbf{Adv}^{\text{kdm}}_{\mathsf{SE}}(\mathcal{A}_{\text{se}}).$$

Proof. The adversaries $\mathcal{A}_{\text{st}}, \mathcal{A}_{\text{se}}$ are given in Fig. 3.15. Notice that both adversaries run $\mathcal{A}$ and can interpret the f_0 provided by it during the setup phase as $L, \{g_\ell\}_{\ell \in L}$ because we $\mathcal{A}$ produced $\mathcal{F}_{\text{ff}}$ functions (as described in Section 3.6). They then construct array $\mathbf{A}$, with $\mathcal{A}_{\text{se}}$ following $\mathsf{Aye}_{\text{ff}}.\mathsf{Enc}$ but using its ENC oracle to perform encryption. $\mathcal{A}_{\text{st}}$ does the same but selects random strings of the appropriate lengths in place of encryption. Notice that $\mathcal{A}_{\text{st}}$ can play $\text{G}^{\text{ss}}_{\mathsf{Aye},\mathcal{L},\mathcal{S}}$ games because the array $\mathbf{A}$ and its queries x are not key-dependent. This follows from the definition of $\mathcal{F}_{\text{ff}}$ functions.

Now consider the hybrid games $\text{G}_0, \text{G}_1, \text{G}_2$. G_0 is exactly what happens in the "real world" in $\text{G}^{\mathcal{F}_{\text{ff}}\text{-ss}}_{\mathsf{Aye}_{\text{ff}},\mathcal{L}_{\text{a}},\mathcal{S}}(\mathcal{A})$ while G_2 is exactly what happens in the "ideal world". This gives us

$$\mathbf{Adv}^{\mathcal{F}_{\text{ff}}\text{-ss}}_{\mathsf{Aye}_{\text{ff}},\mathcal{L}_{\text{a}},\mathcal{S}}(\mathcal{A}) = \Pr[\text{G}_0] - \Pr[\text{G}_2].$$

G_1 is the same as G_0 except that the entries in $\mathbf{A}$ are randomly selected strings of the appropriate length instead of encryptions of the $g_\ell(K^{\text{a}})$. This gives us the following (where b is the challenge bit in the respective games), which together with the above equation completes the proof:

$$\Pr[\text{G}_0] - \Pr[\text{G}_1] = \Pr[\text{G}^{\text{kdm}}_{\mathsf{SE}}(\mathcal{A}_{\text{se}})|b=1] - \Pr[\text{G}^{\text{kdm}}_{\mathsf{SE}}(\mathcal{A}_{\text{se}})|b=0] = \mathbf{Adv}^{\text{kdm}}_{\mathsf{SE}}(\mathcal{A}_{\text{se}})$$
$$\Pr[\text{G}_1] - \Pr[\text{G}_2] = \Pr[\text{G}^{\text{ss}}_{\mathsf{Aye},\mathcal{L},\mathcal{S}}(\mathcal{A}_{\text{st}})|b=1] - \Pr[\text{G}^{\text{ss}}_{\mathsf{Aye},\mathcal{L},\mathcal{S}}(\mathcal{A}_{\text{st}})|b=0] = \mathbf{Adv}^{\text{ss}}_{\mathsf{Aye},\mathcal{L},\mathcal{S}}(\mathcal{A}_{\text{st}})$$

□

Note that Aye_{ff} is a correct and $\mathcal{F}_{\text{ff}}$-secure RH AYE no matter Aye's response type. Also, when Aye is instantiated using standard techniques from the literature (including the scheme in Fig. 3.2) each $\mathsf{Aye}[\ell]$ will be symmetrically encrypted twice. Therefore, we recommend that **FfT**

be used with a RR variant of Aye so that the outer layer of (non-KDM) symmetric encryption will be removed by the server thereby reducing the client's computational overhead from adopting $\mathsf{Aye}_{\mathrm{ff}}$.

It is worth noting here what makes $\mathcal{F}_{\mathrm{ff}}$-security efficiently possible for RH AYE. If we consider the adversaries above which break $\mathcal{F}_{\mathrm{all}}$-security, it is the abuse of key-dependent queries and length information, both of which are disallowed in the definition of $\mathcal{F}_{\mathrm{ff}}$. Based on previous results, it makes sense this is the barrier for efficient key-dependent RH AYE schemes. Since KDM security can be secure for fixed length functions, and query equality pattern seems to be an efficiency barrier for StE [113]. There could be more restricted function classes which capture different efficiency barriers for RH AYE. Capturing other natural barriers aside from fixed format functions remains an open question.

Dedicated RH transform for $\mathcal{F}_{\mathrm{ff}}$-secure AYE.

A natural follow-up question to the above point is: Can we completely avoid encrypting the **A**'s values twice? Intuitively, this should be possible since the layer of KDM-secure SE should be sufficient for data-privacy so any IND$-secure SE used in Aye is not needed. This would further reduce computational overhead during setup and evaluation compared to the generic solution.

We concretize this intuition by stating and proving that the RH AYE variant of CJJ+'s SSE scheme (recalled in Fig. 3.2 as $\mathsf{Aye}_{\mathrm{h}}^{\pi}$) is $\mathcal{F}_{\mathrm{ff}}$-secure, without modification, so long as the symmetric encryption primitive used within is KDM-secure. The proof of this result is very similar to that of $\mathsf{Aye}_{\mathrm{h}}^{\pi}$'s security in the standard model (which has been studied and detailed by CJJ+ and JT [48, 85]) so we omit it for brevity.

Theorem 25 *Let $\mathcal{A}$ be an adversary and $\mathsf{Aye}_{\mathrm{h}}^{\pi}$ be the RH AYE scheme defined in Fig. 3.2 with primitives SE, F leakage algorithm $\mathcal{L}_{\mathrm{h}}^{\pi}$ and simulator $\mathcal{S}_{\mathrm{h}}^{\pi}$. Then there exists $\mathcal{A}_{\mathrm{se}}, \mathcal{A}_{\mathrm{f}}$ such that:*

$$\mathbf{Adv}^{\mathcal{F}_{\mathrm{ff}}\text{-ss}}_{\mathsf{Aye}_{\mathrm{h}}^{\pi},\mathcal{L}_{\mathrm{h}}^{\pi},\mathcal{S}_{\mathrm{h}}^{\pi}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathrm{kdm}}_{\mathsf{SE}}(\mathcal{A}_{\mathrm{se}}) + N \cdot \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(\mathcal{A}_{\mathrm{f}})$$

where N is the maximum number of labels in the array provided by $\mathcal{A}$.

3.6.3 KD-security of Response-Flexible AYE

We now have results on $\mathcal{F}_{\text{ff}}$-security for RH and RR, so a natural follow-up is how this applies to RF AYE. Intuitively, much like with RR AYE we cannot allow the adversary to store key dependent values in any $\mathbf{A}[1\|\ell]$ but should allow arbitrary key-dependent strings in the $\mathbf{A}[0\|\ell]$ entries.

Response-flexible function class.

We formalize this by defining $\mathcal{F}_{\text{rf}}$ for Aye_{f} which is a subclass of $\mathcal{F}_{\text{ff}}$.

We say that $f = (f_0, f_1, \ldots, f_n)$ is in $\mathcal{F}_{\text{rf}}$ if, when f_0 is expressed using per-label functions $\{g_\ell\}_{\ell \in L}$, all $\ell \in L$ where $\ell = 1\|\ell'$ are such that $g_\ell(K_1) = g_\ell(K_2)$ for all $K_1, K_2 \in \mathsf{Aye}_{\text{f}}.\mathsf{KS}$.

$\mathcal{F}_{\text{rf}}$-secure RF AYE.

We can realize such a scheme via the **RfT** transform from Section 3.5 in Fig. 3.4, assuming that SE is KDM-secure. We omit a proof of this result due to its similarity to Theorem 20 and Theorem 24.

Theorem 26 *Let $\mathcal{A}$ be an adversary, $\mathcal{L}$ a leakage algorithm and $\mathcal{S}$ a simulator. Let $\mathsf{Aye}_{\text{f}} = \mathbf{RfT}[\mathsf{Aye}_{\text{r}}, \mathsf{SE}, \mathsf{F}], \mathcal{L}_{\text{f}}$ be as defined in Section 3.5. Then there exists $\mathcal{A}_{\text{ss}}, \mathcal{A}_{\text{se}}, \mathcal{A}_{\text{f}}, \mathcal{S}_{\text{f}}$ such that:*

$$\mathbf{Adv}^{\mathcal{F}_{\text{rf}}\text{-ss}}_{\mathsf{Aye}_{\text{f}},\mathcal{L}_{\text{f}},\mathcal{S}_{\text{f}}}(\mathcal{A}) \leq \mathbf{Adv}^{\text{ss}}_{\mathsf{Aye}_{\text{r}},\mathcal{L},\mathcal{S}}(\mathcal{A}_{\text{ss}}) + \mathbf{Adv}^{\text{kdm}}_{\mathsf{SE}}(\mathcal{A}_{\text{se}}) + M \cdot \mathbf{Adv}^{\text{prf}}_{\mathsf{F}}(\mathcal{A}_{\text{f}}),$$

where M is the maximum number of response-hiding TOK *queries made by $\mathcal{A}$.*

3.7 Acknowledgements

We thank Mihir Bellare, Wei Dai and Keegan Ryan for discussions and insights.

This chapter, in full, is currently being prepared for publication of the material. Cash, David; Hoover, Alexander; Ng, Ruth. The dissertation author was the primary investigator and author of this paper.

Bibliography

[1] CakePHP: Using the IV as the key. http://www.cryptofails.com/post/70059594911/cakephp-using-the-iv-as-the-key. Accessed: 2019-02-12.

[2] encrypted-bigquery-client. https://github.com/google/encrypted-bigquery-client, 2015.

[3] City of chicago data portal. https://data.cityofchicago.org/, 2021.

[4] Sakila sample database. https://dev.mysql.com/doc/sakila/en/, 2021.

[5] Farzaneh Abed, Scott R. Fluhrer, Christian Forler, Eik List, Stefan Lucks, David A. McGrew, and Jakob Wenzel. Pipelineable on-line encryption. In Carlos Cid and Christian Rechberger, editors, *FSE 2014*, volume 8540 of *LNCS*, pages 205–223. Springer, Heidelberg, March 2015.

[6] Tolga Acar, Mira Belenkiy, Mihir Bellare, and David Cash. Cryptographic agility and its relation to circular encryption. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 403–422. Springer, Heidelberg, May / June 2010.

[7] Ghous Amjad, Seny Kamara, and Tarik Moataz. Breach-resistant structured encryption. Cryptology ePrint Archive, Report 2018/195, 2018. https://eprint.iacr.org/2018/195.

[8] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. How to securely release unverified plaintext in authenticated encryption. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 105–125. Springer, Heidelberg, December 2014.

[9] Panagiotis Antonopoulos, Arvind Arasu, Kunal D Singh, Ken Eguro, Nitish Gupta, Rajat Jain, Raghav Kaushik, Hanuma Kodavalla, Donald Kossmann, Nikolas Ogg, et al. Azure sql database always encrypted. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1511–1525, 2020.

[10] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 595–618. Springer, Heidelberg, August 2009.

[11] Gilad Asharov, Moni Naor, Gil Segev, and Ido Shahaf. Searchable symmetric encryption: optimal locality in linear space via two-dimensional balanced allocations. In Daniel Wichs and Yishay Mansour, editors, *48th ACM STOC*, pages 1101–1114. ACM Press, June 2016.

[12] Gilad Asharov, Gil Segev, and Ido Shahaf. Tight tradeoffs in searchable symmetric encryption. Cryptology ePrint Archive, Report 2018/507, 2018. https://eprint.iacr.org/2018/507.

[13] Tomer Ashur, Orr Dunkelman, and Atul Luykx. Boosting authenticated encryption robustness with minimal modifications. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 3–33. Springer, Heidelberg, August 2017.

[14] J. Aumasson, S. Babbage, D.J. Bernstein, C. Cid, J. Daemen, O. Dunkelman, K. Gaj, S. Gueron, P. Junod, A. Langley, D. McGrew, K. Paterson, B. Preneel, C. Rechberger, V. Rijmen, M. Robshaw, P. Sarkar, P. Schaumont, A. Shamir, and I. Verbauwhede. CHAE: Challenges in authenticated encryption. ECRYPT-CSA D1.1, Revision 1.05, March 2017. https://chae.cr.yp.to/whitepaper.html.

[15] Michael Backes, Markus Dürmuth, and Dominique Unruh. OAEP is secure under key-dependent messages. In Josef Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 506–523. Springer, Heidelberg, December 2008.

[16] Sumeet Bajaj and Radu Sion. Trusteddb: A trusted hardware-based database with privacy and data confidentiality. *IEEE Transactions on Knowledge and Data Engineering*, 26(3):752–765, 2013.

[17] Manuel Barbosa and Pooya Farshim. Indifferentiable authenticated encryption. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 187–220. Springer, Heidelberg, August 2018.

[18] Guy Barwell, Daniel P. Martin, Elisabeth Oswald, and Martijn Stam. Authenticated encryption in the face of protocol and side channel leakage. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 693–723. Springer, Heidelberg, December 2017.

[19] Guy Barwell, Daniel Page, and Martijn Stam. Rogue decryption failures: Reconciling AE robustness notions. In Jens Groth, editor, *15th IMA International Conference on Cryptography and Coding*, volume 9496 of *LNCS*, pages 94–111. Springer, Heidelberg, December 2015.

[20] Johes Bater, Gregory Elliott, Craig Eggen, Satyender Goel, Abel Kho, and Jennie Rogers. Smcql: secure querying for federated databases. *Proceedings of the VLDB Endowment*, 10(6):673–684, 2017.

[21] Mihir Bellare, Alexandra Boldyreva, and Silvio Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 259–274. Springer, Heidelberg, May 2000.

[22] Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. Deterministic and efficiently searchable encryption. Cryptology ePrint Archive, Report 2006/186, 2006. http://eprint.iacr.org/2006/186.

[23] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Pseudorandom functions revisited: The cascade construction and its concrete security. In *37th FOCS*, pages 514–523. IEEE Computer Society Press, October 1996.

[24] Mihir Bellare, Anand Desai, Eric Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *38th FOCS*, pages 394–403. IEEE Computer Society Press, October 1997.

[25] Mihir Bellare and Sriram Keelveedhi. Authenticated and misuse-resistant encryption of key-dependent data. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 610–629. Springer, Heidelberg, August 2011.

[26] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Tatsuaki Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 531–545. Springer, Heidelberg, December 2000.

[27] Mihir Bellare, Ruth Ng, and Björn Tackmann. Nonces are noticed: AEAD revisited. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 235–265. Springer, Heidelberg, August 2019.

[28] Mihir Bellare and Phillip Rogaway. On the construction of variable-input-length ciphers. In Lars R. Knudsen, editor, *FSE'99*, volume 1636 of *LNCS*, pages 231–244. Springer, Heidelberg, March 1999.

[29] Mihir Bellare and Phillip Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography. In Tatsuaki Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 317–330. Springer, Heidelberg, December 2000.

[30] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, Heidelberg, May / June 2006.

[31] Mihir Bellare and Björn Tackmann. The multi-user security of authenticated encryption: AES-GCM in TLS 1.3. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 247–276. Springer, Heidelberg, August 2016.

[32] Daniel J. Bernstein. Re: secret message numbers. Message in Google group on cryptographic competitions, October 2013. https://groups.google.com/d/msg/crypto-competitions/n5ECGwYr6Vk/bsEfPWqSAU4J.

[33] D.J. Bernstein. CAESAR call for submissions, final (2014.01.27), 2014.

[34] Francesco Berti, Chun Guo, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. Tedt, a leakage-resilient aead mode for high (physical) security applications. Cryptology ePrint Archive, Report 2019/137, 2019. https://eprint.iacr.org/2019/137.

[35] Vincent Bindschaedler, Paul Grubbs, David Cash, Thomas Ristenpart, and Vitaly Shmatikov. The tao of inference in privacy-protected databases. *Proceedings of the VLDB Endowment*, 11(11):1715–1728, 2018.

[36] J. Black, P. Rogaway, and T. Shrimpton. Encryption-scheme security in the presence of key-dependent messages. Cryptology ePrint Archive, Report 2002/100, 2002. http://eprint.iacr.org/2002/100.

[37] Laura Blackstone, Seny Kamara, and Tarik Moataz. Revisiting leakage abuse attacks. Cryptology ePrint Archive, Report 2019/1175, 2019. https://eprint.iacr.org/2019/1175.

[38] Dan Boneh, Shai Halevi, Michael Hamburg, and Rafail Ostrovsky. Circular-secure encryption from decision Diffie-Hellman. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 108–125. Springer, Heidelberg, August 2008.

[39] Priyanka Bose, Viet Tung Hoang, and Stefano Tessaro. Revisiting AES-GCM-SIV: Multi-user security, faster key derivation, and better bounds. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 468–499. Springer, Heidelberg, April / May 2018.

[40] Raphael Bost. Sophos - forward secure searchable encryption. Cryptology ePrint Archive, Report 2016/728, 2016. http://eprint.iacr.org/2016/728.

[41] Raphaël Bost, Brice Minaud, and Olga Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1465–1482. ACM Press, October / November 2017.

[42] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 505–524. Springer, Heidelberg, August 2011.

[43] Nicolas Bruno and Luis Gravano. *Statistics on query expressions in relational database management systems*. PhD thesis, Columbia University, 2003.

[44] CAESAR Committee. Cryptographic competitions: Caesar call for submissions, final (2014.01.27). https://competitions.cr.yp.to/caesar-call.html. Accessed: 2018-07-23.

[45] Yang Cao, Wenfei Fan, Yanghao Wang, and Ke Yi. Querying shared data with security heterogeneity. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 575–585, 2020.

[46] David Cash, Matthew Green, and Susan Hohenberger. New definitions and separations for circular security. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 540–557. Springer, Heidelberg, May 2012.

[47] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 668–679. ACM Press, October 2015.

[48] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *NDSS 2014*. The Internet Society, February 2014.

[49] David Cash, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for Boolean queries. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 353–373. Springer, Heidelberg, August 2013.

[50] David Cash and Stefano Tessaro. The locality of searchable symmetric encryption. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 351–368. Springer, Heidelberg, May 2014.

[51] Ashok K Chandra and Philip M Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 77–90, 1977.

[52] Melissa Chase and Seny Kamara. Structured encryption and controlled disclosure. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 577–594. Springer, Heidelberg, December 2010.

[53] Sherman SM Chow, Jie-Han Lee, and Lakshminarayanan Subramanian. Two-party computation model for privacy-preserving queries over distributed databases. In *NDSS*. Citeseer, 2009.

[54] Valentina Ciriani, Sabrina De Capitani Di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Keep a few: Outsourcing data while maintaining confidentiality. In *European Symposium on Research in Computer Security*, pages 440–455. Springer, 2009.

[55] Aisling Connolly, Pooya Farshim, and Georg Fuchsbauer. Security of symmetric primitives against key-correlated attacks. *IACR Transactions on Symmetric Cryptology*, pages 193–230, 2019.

[56] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 79–88. ACM Press, October / November 2006.

[57] Ernesto Damiani, S De Capitani Vimercati, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Balancing confidentiality and efficiency in untrusted relational dbmss. In *Proceedings of the 10th ACM conference on Computer and communications security*, pages 93–102, 2003.

[58] Ioannis Demertzis, Dimitrios Papadopoulos, and Charalampos Papamanthou. Searchable encryption with optimal locality: Achieving sublogarithmic read efficiency. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 371–406. Springer, Heidelberg, August 2018.

[59] Ioannis Demertzis, Dimitrios Papadopoulos, Charalampos Papamanthou, and Saurabh Shintre. Seal: Attack mitigation for encrypted databases via adjustable leakage. In *29th USENIX Security Symposium (USENIX Security 20)*, 2020.

[60] Yevgeniy Dodis, Paul Grubbs, Thomas Ristenpart, and Joanne Woodage. Fast message franking: From invisible salamanders to encryptment. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 155–186. Springer, Heidelberg, August 2018.

[61] Morris Dworkin. Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC. NIST Special Publication 800-38D, November 2007.

[62] Sergei Evdokimov and Oliver Günther. Encryption techniques for secure database outsourcing. In *European Symposium on Research in Computer Security*, pages 327–342. Springer, 2007.

[63] Sky Faber, Stanislaw Jarecki, Hugo Krawczyk, Quan Nguyen, Marcel Rosu, and Michael Steiner. Rich queries on encrypted data: Beyond exact matches. In *European symposium on research in computer security*, pages 123–145. Springer, 2015.

[64] Pooya Farshim, Claudio Orlandi, and Răzvan Roşie. Security of symmetric primitives under incorrect usage of keys. *IACR Trans. Symm. Cryptol.*, 2017(1):449–473, 2017.

[65] Ewan Fleischmann, Christian Forler, and Stefan Lucks. McOE: A family of almost foolproof on-line authenticated encryption schemes. In Anne Canteaut, editor, *FSE 2012*, volume 7549 of *LNCS*, pages 196–215. Springer, Heidelberg, March 2012.

[66] Sanjam Garg, Payman Mohassel, and Charalampos Papamanthou. Tworam: efficient oblivious ram in two rounds with applications to searchable encryption. In *Annual International Cryptology Conference*, pages 563–592. Springer, 2016.

[67] Eu-Jin Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003. http://eprint.iacr.org/2003/216.

[68] Rishab Goyal, Venkata Koppula, and Brent Waters. Separating IND-CPA and circular security for unbounded length key cycles. In Serge Fehr, editor, *PKC 2017, Part I*, volume 10174 of *LNCS*, pages 232–246. Springer, Heidelberg, March 2017.

[69] Rishab Goyal, Venkata Koppula, and Brent Waters. Separating semantic and circular security for symmetric-key bit encryption from the learning with errors assumption. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 528–557. Springer, Heidelberg, April / May 2017.

[70] Patrick Grofig, Isabelle Hang, Martin Härterich, Florian Kerschbaum, Mathias Kohler, Andreas Schaad, Axel Schröpfer, and Walter Tighzert. Privacy by encrypted databases. In *Privacy Technologies and Policy - Second Annual Privacy Forum, APF 2014, Athens, Greece, May 20-21, 2014. Proceedings*, pages 56–69, 2014.

[71] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G Paterson. Pump up the volume: Practical database reconstruction from volume leakage on range queries. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 315–331, 2018.

[72] Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. Message franking via committing authenticated encryption. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 66–97. Springer, Heidelberg, August 2017.

[73] Paul Grubbs, Kevin Sekniqi, Vincent Bindschaedler, Muhammad Naveed, and Thomas Ristenpart. Leakage-abuse attacks against order-revealing encryption. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 655–672. IEEE, 2017.

[74] Shay Gueron, Adam Langley, and Yehuda Lindell. AES-GCM-SIV: Specification and analysis. Cryptology ePrint Archive, Report 2017/168, 2017. http://eprint.iacr.org/2017/168.

[75] Shay Gueron and Yehuda Lindell. GCM-SIV: Full nonce misuse-resistant authenticated encryption at under one cycle per byte. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 109–119. ACM Press, October 2015.

[76] Zichen Gui, Oliver Johnson, and Bogdan Warinschi. Encrypted databases: New volume attacks against range queries. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 361–378, 2019.

[77] Hakan Hacigümüş, Bala Iyer, Chen Li, and Sharad Mehrotra. Executing sql over encrypted data in the database-service-provider model. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 216–227, 2002.

[78] Timon Hackenjos, Florian Hahn, and Florian Kerschbaum. Sagma: Secure aggregation grouped by multiple attributes. *ACM SIGMOD Record*, 2020.

[79] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J Alex Halderman. Mining your ps and qs: Detection of widespread weak keys in network devices. In *USENIX Security Symposium*, volume 8, page 1, 2012.

[80] Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway. Robust authenticated-encryption AEZ and the problem that it solves. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 15–44. Springer, Heidelberg, April 2015.

[81] Viet Tung Hoang, Reza Reyhanitabar, Phillip Rogaway, and Damian Vizár. Online authenticated-encryption and its nonce-reuse misuse-resistance. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 493–517. Springer, Heidelberg, August 2015.

[82] Viet Tung Hoang, Stefano Tessaro, and Aishwarya Thiruvengadam. The multi-user security of GCM, revisited: Tight bounds for nonce randomization. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1429–1440. ACM Press, October 2018.

[83] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *NDSS 2012*. The Internet Society, February 2012.

[84] Tetsu Iwata, Keisuke Ohashi, and Kazuhiko Minematsu. Breaking and repairing GCM security proofs. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 31–49. Springer, Heidelberg, August 2012.

[85] Joseph Jaeger and Nirvan Tyagi. Handling adaptive compromise for practical encryption schemes. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 3–32. Springer, Heidelberg, August 2020.

[86] Antoine Joux. Authentication failures in NIST version of GCM, 2006. Comments submitted to NIST modes of operation process, https://csrc.nist.gov/csrc/media/projects/block-cipher-techniques/documents/bcm/comments/800-38-series-drafts/gcm/joux_comments.pdf.

[87] Seny Kamara and Tarik Moataz. Boolean searchable symmetric encryption with worst-case sub-linear complexity. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 94–124. Springer, Heidelberg, April / May 2017.

[88] Seny Kamara and Tarik Moataz. Encrypted multi-maps with computationally-secure leakage. *IACR Cryptol. ePrint Arch.*, 2018:978, 2018.

[89] Seny Kamara and Tarik Moataz. SQL on structurally-encrypted databases. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part I*, volume 11272 of *LNCS*, pages 149–180. Springer, Heidelberg, December 2018.

[90] Seny Kamara and Tarik Moataz. Computationally volume-hiding structured encryption. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 183–213. Springer, Heidelberg, May 2019.

[91] Seny Kamara, Tarik Moataz, and Olga Ohrimenko. Structured encryption and leakage suppression. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 339–370. Springer, Heidelberg, August 2018.

[92] Seny Kamara, Tarik Moataz, Stan Zdonik, and Zheguang Zhao. An optimal relational database encryption scheme. Cryptology ePrint Archive, Report 2020/274, 2020. https://eprint.iacr.org/2020/274 Accessed: 2020-02-29.

[93] Seny Kamara and Charalampos Papamanthou. Parallel and dynamic searchable symmetric encryption. In *International conference on financial cryptography and data security*, pages 258–274. Springer, 2013.

[94] Seny Kamara, Charalampos Papamanthou, and Tom Roeder. Dynamic searchable symmetric encryption. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012*, pages 965–976. ACM Press, October 2012.

[95] Murat Kantarcıoglu and Chris Clifton. Security issues in querying encrypted data. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 325–337. Springer, 2005.

[96] Jonathan Katz and Moti Yung. Unforgeable encryption and chosen ciphertext secure modes of operation. In Bruce Schneier, editor, *FSE 2000*, volume 1978 of *LNCS*, pages 284–299. Springer, Heidelberg, April 2001.

[97] Venkata Koppula and Brent Waters. Circular security separations for arbitrary length cycles from LWE. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 681–700. Springer, Heidelberg, August 2016.

[98] Hugo Krawczyk. LFSR-based hashing and authentication. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 129–139. Springer, Heidelberg, August 1994.

[99] Ted Krovetz and Phillip Rogaway. The software performance of authenticated-encryption modes. In Antoine Joux, editor, *FSE 2011*, volume 6733 of *LNCS*, pages 306–327. Springer, Heidelberg, February 2011.

[100] Kaoru Kurosawa and Tetsu Iwata. TMAC: Two-key CBC MAC. In Marc Joye, editor, *CT-RSA 2003*, volume 2612 of *LNCS*, pages 33–49. Springer, Heidelberg, April 2003.

[101] Encrypted Systems Lab. The clusion library. https://github.com/encryptedsystems/Clusion, 2020.

[102] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable block ciphers. *Journal of Cryptology*, 24(3):588–613, July 2011.

[103] Atul Luykx, Bart Mennink, and Kenneth G. Paterson. Analyzing multi-key security degradation. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 575–605. Springer, Heidelberg, December 2017.

[104] David McGrew. An interface and algorithms for authenticated encryption. IETF Network Working Group, RFC 5116, January 2008.

[105] David A. McGrew and John Viega. The security and performance of the Galois/counter mode (GCM) of operation. In Anne Canteaut and Kapalee Viswanathan, editors, *INDOCRYPT 2004*, volume 3348 of *LNCS*, pages 343–355. Springer, Heidelberg, December 2004.

[106] Carl H Meyer and Stephen M Matyas. *CRYPTOGRAPHY: A new dimension in computer data security: A guide for the design and implementation of secure systems*. Wiley, 1982.

[107] Kazuhiko Minematsu. Authenticated encryption with small stretch (or, how to accelerate AERO). In Joseph K. Liu and Ron Steinfeld, editors, *ACISP 16, Part II*, volume 9723 of *LNCS*, pages 347–362. Springer, Heidelberg, July 2016.

[108] Chanathip Namprempre, Phillip Rogaway, and Thomas Shrimpton. Reconsidering generic composition. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 257–274. Springer, Heidelberg, May 2014.

[109] Chanathip Namprempre, Phillip Rogaway, and Tom Shrimpton. AE5 security notions: Definitions implicit in the CAESAR call. Cryptology ePrint Archive, Report 2013/242, 2013. http://eprint.iacr.org/2013/242.

[110] Muhammad Naveed, Seny Kamara, and Charles V. Wright. Inference attacks on property-preserving encrypted databases. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 644–655. ACM Press, October 2015.

[111] Muhammad Naveed, Manoj Prabhakaran, and Carl A. Gunter. Dynamic searchable encryption via blind storage. In *2014 IEEE Symposium on Security and Privacy*, pages 639–654. IEEE Computer Society Press, May 2014.

[112] Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. Private stateful information retrieval. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1002–1019. ACM Press, October 2018.

[113] Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. Lower bounds for encrypted multi-maps and searchable encryption in the leakage cell probe model. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 433–463. Springer, Heidelberg, August 2020.

[114] Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 79–93. ACM Press, November 2019.

[115] Thomas Peyrin and Yannick Seurin. Counter-in-tweak: Authenticated encryption modes for tweakable block ciphers. In Matthew Robshaw and Jonathan Katz, editors,

CRYPTO 2016, Part I, volume 9814 of *LNCS*, pages 33–63. Springer, Heidelberg, August 2016.

[116] Raluca Ada Popa, Catherine MS Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 85–100, 2011.

[117] David Pouliot and Charles V. Wright. The shadow nemesis: Inference attacks on efficiently deployable, efficiently searchable encryption. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1341–1352. ACM Press, October 2016.

[118] Reddit. Hash of message as nonce?, 2015. https://redd.it/3c504m.

[119] Adam Rivkin. Hybrid indexing simulations. https://github.com/AdamRivkin/Hybrid-Indexing-Simulations, 2021.

[120] Phillip Rogaway. Authenticated-encryption with associated-data. In Vijayalakshmi Atluri, editor, *ACM CCS 2002*, pages 98–107. ACM Press, November 2002.

[121] Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In Pil Joong Lee, editor, *ASIACRYPT 2004*, volume 3329 of *LNCS*, pages 16–31. Springer, Heidelberg, December 2004.

[122] Phillip Rogaway. Nonce-based symmetric encryption. In Bimal K. Roy and Willi Meier, editors, *FSE 2004*, volume 3017 of *LNCS*, pages 348–359. Springer, Heidelberg, February 2004.

[123] Phillip Rogaway. The evolution of authenticated encryption. Real World Cryptography Workshop, Stanford, January 2013. https://crypto.stanford.edu/RealWorldCrypto/slides/phil.pdf.

[124] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: A block-cipher mode of operation for efficient authenticated encryption. In Michael K. Reiter and Pierangela Samarati, editors, *ACM CCS 2001*, pages 196–205. ACM Press, November 2001.

[125] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 373–390. Springer, Heidelberg, May / June 2006.

[126] Phillip Rogaway, Mark Wooding, and Haibin Zhang. The security of ciphertext stealing. In Anne Canteaut, editor, *FSE 2012*, volume 7549 of *LNCS*, pages 180–195. Springer, Heidelberg, March 2012.

[127] Joseph Sack. Optimizing your query plans with the sql server 2014 cardinality estimator, 2014.

[128] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy*, pages 44–55. IEEE Computer Society Press, May 2000.

[129] Emil Stefanov, Charalampos Papamanthou, and Elaine Shi. Practical dynamic searchable encryption with small leakage. In *NDSS 2014*. The Internet Society, February 2014.

[130] Stephen Lyle Tu, M Frans Kaashoek, Samuel R Madden, and Nickolai Zeldovich. Processing analytical queries over encrypted data. 2013.

[131] Serge Vaudenay and Damian Vizár. Under pressure: Security of caesar candidates beyond their guarantees. Cryptology ePrint Archive, Report 2017/1147, 2017. https://eprint.iacr.org/2017/1147.

[132] Cong Wang, Ning Cao, Jin Li, Kui Ren, and Wenjing Lou. Secure ranked keyword search over encrypted cloud data. In *2010 IEEE 30th international conference on distributed computing systems*, pages 253–262. IEEE, 2010.

[133] Hongjun Wu and Bart Preneel. AEGIS: A fast authenticated encryption algorithm. In Tanja Lange, Kristin Lauter, and Petr Lisonek, editors, *SAC 2013*, volume 8282 of *LNCS*, pages 185–201. Springer, Heidelberg, August 2014.

[134] Zhiqiang Yang, Sheng Zhong, and Rebecca N Wright. Privacy-preserving queries on encrypted data. In *European Symposium on Research in Computer Security*, pages 479–495. Springer, 2006.

[135] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 707–720. USENIX Association, August 2016.