Title
Protocols and security proofs for data authentication

Permalink
https://escholarship.org/uc/item/767372fz

Author
Mityagin, Anton

Publication Date
2006-01-01

Peer reviewed|Thesis/dissertation
Protocols and Security Proofs for Data Authentication

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

by

Anton Mityagin

Committee in charge:

Professor Mihir Bellare, Chair
Professor Samuel Buss
Professor Russell Impagliazzo
Professor Daniele Micciancio
Professor Alon Orlitsky

2006
Copyright
Anton Mityagin, 2006
All rights reserved.
The dissertation of Anton Mityagin is approved, and it is acceptable in quality and form for publication on microfilm:

Chair

University of California, San Diego

2006
# TABLE OF CONTENTS

Signature Page ........................................ iii
Table of Contents ....................................... iv
List of Figures .......................................... vii
List of Tables .......................................... viii
Acknowledgements ..................................... ix
Vita and Publications .................................. x
Abstract ................................................ xi

1 Introduction ......................................... 1
  1.1 Data Authentication Primitives .................. 1
    1.1.1 Security of Message Authentication Schemes .... 1
    1.1.2 Negative Results for Message Authentication .... 2
    1.1.3 Negative Results for Authenticated Encryption ... 5
    1.1.4 Data Authentication Primitive and the Equivalence Result ... 6
    1.1.5 Repair: Recovering Security of Practical MA Schemes ... 8
    1.1.6 Quality of the Equivalence Reduction ............ 9
    1.1.7 Concrete Security Improvements .................. 10
    1.1.8 Remarks ........................................ 12
  1.2 Append-Only Signatures ......................... 13
    1.2.1 Definition ...................................... 13
    1.2.2 Constructions ................................... 14
    1.2.3 Relation to Hierarchical Identity-Based Signatures ... 15
    1.2.4 Related Work ................................... 16
    1.2.5 Application to Secure Routing .................. 18

2 Security of MA and AE Schemes against Multiple Verification Queries ... 19
  2.1 Definitions ....................................... 19
  2.2 Template of a Reduction ........................ 22
  2.3 MA-UF Security against Single Verification Query doesn’t Imply Security against Multiple Queries ... 25
  2.4 On the Order of Signing and Verification Queries .... 28
  2.5 Authenticated Encryption (INT-PTXT): Security against Single Verification Query Doesn’t Imply Security against Multiple Queries ... 31
    2.5.1 Definitions .................................... 31
    2.5.2 Results ....................................... 33
  2.6 Acknowledgements ................................ 34
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Definitions</td>
<td>35</td>
</tr>
<tr>
<td>3.2</td>
<td>Data Authentication Primitives for Existing Protocols</td>
<td>39</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Message Authentication (MA) Schemes</td>
<td>39</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Authenticated Encryption (AE) Schemes</td>
<td>40</td>
</tr>
<tr>
<td>3.2.3</td>
<td>XOR-tag Schemes</td>
<td>41</td>
</tr>
<tr>
<td>3.2.4</td>
<td>Nonce-Based MA Schemes</td>
<td>43</td>
</tr>
<tr>
<td>3.2.5</td>
<td>Authenticated Encryption with Associated Data (AEAD)</td>
<td>44</td>
</tr>
<tr>
<td>3.3</td>
<td>Any Revealing Function Is Equivalent to Trivial</td>
<td>45</td>
</tr>
<tr>
<td>3.4</td>
<td>Security of Data Authentication Primitives against Multiple Verification Queries</td>
<td>46</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Security under One Query Implies Security under Multiple Queries</td>
<td>46</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Improving Concrete Security</td>
<td>51</td>
</tr>
<tr>
<td>3.5</td>
<td>Reduction Tightness</td>
<td>53</td>
</tr>
<tr>
<td>3.6</td>
<td>Establishing Security of Practical Protocols under Multiple Verification Queries</td>
<td>56</td>
</tr>
<tr>
<td>3.6.1</td>
<td>Classes of MA-SUF Secure MA Schemes</td>
<td>56</td>
</tr>
<tr>
<td>3.6.2</td>
<td>Improvements for PRF Based MACs</td>
<td>58</td>
</tr>
<tr>
<td>3.6.3</td>
<td>Generalized Carter-Wegman MACs</td>
<td>60</td>
</tr>
<tr>
<td>3.6.4</td>
<td>EAX Mode of Operation</td>
<td>62</td>
</tr>
<tr>
<td>3.7</td>
<td>Acknowledgements</td>
<td>64</td>
</tr>
<tr>
<td>4.1</td>
<td>Definitions</td>
<td>66</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Append-only Signatures</td>
<td>66</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Public Key Signature Schemes</td>
<td>69</td>
</tr>
<tr>
<td>4.2</td>
<td>Efficient AOS Constructions</td>
<td>71</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Certificate-Based Append-Only Signatures</td>
<td>71</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Security Proof for Certificate-Based AOS (Theorem 4.2.1)</td>
<td>73</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Shorter Signatures via Aggregation</td>
<td>78</td>
</tr>
<tr>
<td>4.2.4</td>
<td>Compact Signatures via the Boneh-Goh-Boyen HIBE</td>
<td>79</td>
</tr>
<tr>
<td>4.2.5</td>
<td>AOS via Hash Trees</td>
<td>83</td>
</tr>
<tr>
<td>4.2.6</td>
<td>Security Proof for Hash Tree AOS (Theorem 4.2.5)</td>
<td>87</td>
</tr>
<tr>
<td>4.2.7</td>
<td>AOS via One-time Signatures</td>
<td>92</td>
</tr>
<tr>
<td>4.3</td>
<td>Relations between HIBS and AOS</td>
<td>93</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Definition of HIBS</td>
<td>95</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Constructing AOS from HIBS</td>
<td>97</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Constructing HIBS from AOS</td>
<td>99</td>
</tr>
<tr>
<td>4.3.4</td>
<td>Discussion</td>
<td>102</td>
</tr>
<tr>
<td>4.4</td>
<td>Append-only Message Authentication Schemes</td>
<td>103</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Definitions</td>
<td>103</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Construction of AOMA scheme</td>
<td>105</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

2.1 Security game defining standard (MA-UF) and strong (MA-SUF) security of MA scheme $\Pi = (\text{SGN}, \text{VF})$ with key space $K$. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . …
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Efficiency of certificate-based AOS. Data is given for messages of length ( n ).</td>
<td>73</td>
</tr>
<tr>
<td>4.2</td>
<td>Efficiency of AOS with signature aggregation. Data is given for messages of length ( n ).</td>
<td>79</td>
</tr>
<tr>
<td>4.3</td>
<td>Efficiency of AOS2. ( d ) represents the maximum message length. ( e(\cdot, \cdot) ) is a pairing operation on elements of the group ( G_1 ) as used by the HIBE scheme.</td>
<td>80</td>
</tr>
<tr>
<td>4.4</td>
<td>Efficiency of the hash-tree based scheme AOS3. ( d ) represents the maximum message length, ( n ) represents the length of a given message, and ( k ) is the security parameter.</td>
<td>86</td>
</tr>
<tr>
<td>4.5</td>
<td>Efficiency of ( AOMA1 ). Here ( n ) represents the length of a given message, and ( k ) is the security parameter. ( H(\cdot, \cdot) ) denotes an invocation of a hash function.</td>
<td>106</td>
</tr>
</tbody>
</table>
ACKNOWLEDGEMENTS

I’m very grateful to my advisor Mihir Bellare for his guidance and support during my studies. I also thank my past and present labmates for bringing some fun and excitement to the working process.

Parts of Chapters 2 and 3 of this dissertation appeared in the paper

The dissertation author was the primary investigator and author of this paper.

Parts of Chapter 4 of this dissertation appeared in the paper

The dissertation author was the primary investigator and author of this paper.
VITA

2001  B.S.  Moscow State University
2003  M.S.  Weizmann Institute of Science
2006  Ph.D.  University of California, San Diego

PUBLICATIONS


FIELDS OF STUDY

Major Field: Computer Science

Studies in Cryptography.

Professor Mihir Bellare, University of California, San Diego
ABSTRACT OF THE DISSERTATION

Protocols and Security Proofs for Data Authentication

by

Anton Mityagin

Doctor of Philosophy in Computer Science
University of California, San Diego, 2006
Professor Mihir Bellare, Chair

This thesis studies security of various cryptographic primitives which provide for
data authentication.

We first study how security of existing primitives such as message authentication,
authentication encryption, AEAD or XOR-tag schemes depends on the number of veri-
ification attempts towards forgery, the adversary is able to make.

We point out that, contrary to popular belief, allowing a message authentication
adversary multiple verification attempts towards forgery is not equivalent to allowing it
a single one, so that the notion of security that most message authentication schemes are
proven to meet does not guarantee their security in practice.

We next develop a framework for establishing security of various cryptographic
protocols against multiple verification queries. We introduce a new primitive, called
the data authentication primitive which generalizes message authentication, authenti-
cated encryption and other primitives. We specify a condition under which security of
a data authentication primitive against multiple verification queries is equivalent to that
against a single query and prove security against multiple verification queries for any
data authentication primitives that satisfy to this condition.

We use the results on data authentication primitives to recover security of popular
classes of message authentication schemes such as MACs (including HMAC and PRF-
based MACs) and CW-schemes. As well, we improve concrete security of the EAX
mode of operation and of generalized Carter-Wegman message authentication schemes, where we show that multiple verification queries give virtually no advantage to the adversary.

We also present a new primitive for data authentication—*Append-only Signatures (AOS)*—with the property that any party given an AOS signature on message $M_1$ can “append” this signature with any message $M_2$ to obtain the signature on a concatenation of $M_1$ and $M_2$. We define the security of AOS, present concrete AOS schemes, and prove their security under standard assumptions. In addition, we find that despite its simple definition, AOS is equivalent to Hierarchical Identity-based Signatures (HIBS) through efficient and security-preserving reductions. We finally show how to apply AOS to authenticate route announcements in the *BGP routing protocol*, which is an important open problem in network security.
1 Introduction

1.1 Data Authentication Primitives

1.1.1 Security of Message Authentication Schemes

MESSAGE AUTHENTICATION. A message authentication (MA) scheme allows parties sharing a key $K$ to authenticate data they send to each other. The sender applies a tag generation algorithm $SGN$ to $K$ and the message $M$ to get a tag $T$, and then sends $M, T$ to the receiver. The latter applies a verification algorithm $VF$ to $K$, a received message, and its accompanying tag, to get an output of 1 (accept) or 0 (reject), indicating whether or not the message should be considered authentic.

Message authentication schemes are pervasive in practice. In a typical usage, public-key cryptography is first used to exchange a private key $K$, and the latter is then used to authenticate (and also possibly encrypt) data. This happens in Internet security protocols like SSL (used to secure credit card numbers in electronic commerce), SSH (secure remote login) and IPSEC.

Much work goes into obtaining high-performance, secure MA schemes, and there are a variety of schemes in existence and use, including HMAC [3], block-cipher based MACs [10, 53, 19, 8, 18, 40, 45, 39] and Carter-Wegman (CW) MA schemes [62, 44, 57, 36, 16, 17].

STANDARD SECURITY OF MA SCHEMES AND THE NUMBER OF VERIFICATION QUERIES. The natural definition of security [10] comes by extending the one for digital
signatures [35]. Namely, an adversary, allowed a chosen-message attack (via access to a tagging oracle \( \text{SGN}(K, \cdot) \)), should be unable to produce a message-tag pair \((M, T)\) that is valid (meaning \( \text{VF}(K, M, T) = 1 \)) and for which the message is fresh (meaning, was not queried to the tagging oracle).

This variant of the definition however gives the adversary only one verification attempt. Another variant of this definition [8] gives the adversary, in addition to the tagging oracle, also a verification oracle \( \text{VF}(K, \cdot, \cdot) \), which models multiple forgery attempts by the adversary. The adversary should be unable to make a query \((M, T)\) to its verification oracle such that the latter returns 1 but \(M\) was not previously queried to the tag oracle.

Let us refer to this second variant of the definition as standard unforgeability of MA schemes against multiple verification queries and denote it by \( \text{MA-UF} \). When talking about \( \text{MA-UF} \) security of MA schemes we will usually specify how many verification queries (multiple or a single one) the adversary is allowed to make. Note that the first variant we discussed above is just the special case of the second one in which only a single verification query is allowed (\( \text{MA-UF} \) security against a single verification query).

In practice, it is certainly possible for an adversary to make multiple verification attempts. For example, a server authenticating a client under their common key functions as the verification oracle, and an adversary can forward many transmissions to it. A scheme that achieves \( \text{MA-UF} \) security against a single verification query but not against multiple queries is thus clearly not providing adequate security for practical usage. Examples of attacks in practice that exploit multiple verifications are those on the ANSI retail MAC [50, 54].

1.1.2 Negative Results for Message Authentication

The Conjectured Equivalency Result. It turns out the fact that security against multiple verification queries is the “right” model is quite well understood in the com-
munity and literature. Nonetheless, it is the MA-UF notion under a single verification query that most schemes are proven to meet [10, 53, 3, 19, 18, 16, 17, 45, 39, 40]; Exceptions are [8, 7].

It appears to be due to the belief that MA-UF security under a single verification query implies MA-UF security under multiple queries. More precisely, the belief is that if an adversary \( A \) makes \( q_v \) verification oracle queries, its advantage (probability of forgery) is not more than \( q_v \) times that of an adversary \( B \) of comparable time making just one verification oracle query. So the belief is that although a difference in concrete security does manifest itself, securities against single-query and multiple-queries adversaries are polynomially equivalent. This view is expressed for example in [17, Page 21], who say:

> This definitional choice is pretty inconsequential ... generalizing to \( q_v \geq 1 \) verification oracle queries will increase the adversary’s chance of success by at most \( q_v \). The proof is simple and the observation is well-known, so the proof is omitted.

The “choice” they refer to is whether to allow one or many queries to the verification oracle. The “proof” they refer to does not appear anywhere, but the folklore argument is that \( B \) can guess one of the verification queries of \( A \) to use as its own, single verification query, answering previous ones by “0.”

**The Equivalency Does Not Hold.** In Section 2.2 we show why the above-mentioned folklore argument is incorrect. This leaves open the question of whether or not it can be patched to show the equivalence, but we then go on to show that it cannot. Namely we show in Theorem 2.3.1 that MA-UF security under a single verification query does not imply MA-UF security under multiple queries. We do this by presenting a MA scheme that is MA-UF secure against a single verification queries but not under multiple queries. (Naturally this requires the assumption that some MA-UF secure MA scheme exists, otherwise the question is vacuous.) Thus, contrary to the expectation, the MA-UF security notions against single-query and multiple-queries adversaries are not equivalent.
Intuitively, the problem is malleability [27]. In particular, multiple verification queries can add power when an adversary can modify a valid tag for a message into another, different valid tag for the same message.

**Strong Unforgeability of MA Schemes.** Along the lines of standard unforgeability notion (MA-UF) researchers considered a stronger security notion called strong unforgeability [12] which we denote by MA-SUF. Strong unforgeability means that the adversary is not only unable to forge a tag for a new message, but also unable to forge a new tag for an old (meaning, already tagged) message. The formalization is with respect to adversaries getting both a tagging and a verification oracle as above, and again we consider security against adversaries making only a single verification query (MA-SUF security against a single verification query) and against those making multiple queries (MA-SUF security against multiple verification queries).

Our further results imply that MA-SUF security against a single verification query implies MA-SUF security against multiple verification queries. Thus, in contrast to standard unforgeability, the strong notions of unforgeability against a single and multiple verification queries are equivalent.

**Another Unproven Conjecture on the Order in Which the Verification Queries are Made.** Next, we take a closer look at the security proofs for XOR MAC [8] and generalized Carter-Wegman MAC [7]. Both protocols were shown to be MA-SUF secure MA schemes against multiple-queries adversaries and thus our previous security concerns are not relevant to both of the protocols. However in the security proofs of these protocols the authors make the same claim about the order in which adversaries make signing and verification queries. They claim that in order to prove MA-SUF security of their protocols (against multiple queries) it suffices to establish security against only those adversaries who first make all signing queries and then make all verification queries.

In the case of XOR MAC, a preliminary version of the paper [8] suggested without proof that verification queries could be postponed in their scheme, and the proof of the result given later in the full version [9] did not make this claim but used a different
approach. In the case of generalized Carter-Wegman MAC [7], authors first proved that their scheme is secure if an adversary makes all signing queries prior to making any verification queries and then provided a (very brief) sketch on why this implies security against general adversaries. We weren’t able to complete their sketch and moreover, the best security proof for their scheme we were able to find requires a different approach.

We do not know whether the original statements were true or not, but it motivates us to ask a general question, whether security of a MA scheme against adversaries who make all verification queries after all signing queries implies security against general adversaries (who can interleave signing and verification queries). In Section 2.4 we observe that the answer to this question is negative. We construct a MA scheme, which provably provides strong unforgeability against adversaries who make all verification queries after all signing queries and which can be broken by efficient adversaries who do not observe this behavior.

1.1.3 Negative Results for Authenticated Encryption

Similarly to message authentication schemes we establish negative results for authenticated encryption (AE) schemes [12].

There are two notions of integrity for authenticated encryption: integrity of plaintexts, INT-PTXT (an analog of standard unforgeability for MA schemes) and integrity of ciphertexts, INT-CTXT (an analog of strong unforgeability for AE schemes). In either of the cases we may consider versions with one or multiple verification attempts (here the verification oracle takes a ciphertext and says whether or not it is valid). In the literature, INT-CTXT security against one verification query was considered in [42, 14], and INT-CTXT and INT-PTXT against multiple verification queries in [12]. INT-PTXT against a single verification was not considered prior to our work. Our negative results about MA schemes extend, and in particular we show that INT-PTXT security against a single verification query does not imply INT-PTXT security against multiple queries (Section 2.5) but INT-CTXT security notions against one verification and against multiple queries are equivalent (Theorem 3.4.2).
Our results justify that integrity of plaintexts is the “right” security notion for authenticated encryption.

1.1.4 Data Authentication Primitive and the Equivalence Result

The Data Authentication Primitive. To obtain positive results on security of message authentication schemes against multiple verification queries we develop a framework for establishing security results on various cryptographic protocols that provide data authentication. Since several standard cryptographic primitives provide similar security properties such as data authenticity or unforgeability, the same security results (such as equivalence of security against one and against multiple verification queries) hold for many of them. Our approach allows to avoid proving the same theorems over again for each of the standard primitives but instead to prove them once for all of the primitives.

We introduce a new primitive, called the data authentication primitive which generalizes message authentication, authenticated encryption and other primitives. At a high level, the data authentication primitive (similarly to a MA scheme) consists of signing and verification algorithms. The difference with MA schemes is that the verification algorithm is no longer limited to take input message/tag pairs but can take input arbitrary strings (specified by a particular protocol) which we call by verification messages. Clearly, every MA scheme is also a data authentication primitive where signing and verification algorithms are the same and verification messages are message/tag pairs of the MA scheme. However, there are many other instances of a data authentication primitive, which are not alike MA schemes.

Associated to a data authentication primitive is a security notion called unforgeability (UF). It is a natural generalization of the security notion for MA schemes. Namely, an adversary, allowed a chosen-message attack (via access to a signing oracle $\text{SGN}(K, \cdot)$) with multiple forgery attempts (via access to verification oracle $\text{VF}(K, \cdot)$), should be unable to make a verification query $V$ which is both valid (meaning $\text{VF}(K,$
$V) = 1$) and *fresh* with respect to previous signing queries. Here comes another generality provided by the data authentication primitive. Freshness of a verification message $V$ can be defined by an arbitrary function

$$\text{Fresh}(V, M_1, T_1, \ldots, M_n, T_n),$$

which determines freshness of $V$ with respect to previous signing queries $M_1, \ldots, M_n$ made by the adversary and oracle’s answers $T_1, \ldots, T_n$ to these queries. For example, MA schemes with standard unforgeability have a freshness function

$$\text{Fresh}_{\text{MA-UF}}((M, T), M_1, T_1, \ldots, M_n, T_n),$$

which defines a verification message (which is a message/tag pair $(M, T)$) as fresh if $M$ is different from all of $M_1, \ldots, M_n$.

In Section 3.2 we show how standard cryptographic primitives such as MA schemes (under strong unforgeability), AE schemes (under integrity of ciphertexts) [12], AEAD schemes [57], XOR-tag schemes [15] and nonce-based MA and AE schemes are translated to data authentication primitives, such that security of these primitives is equivalent to security of the corresponding data authentication primitives. These primitives provide forms of data authentication and have different protocol specifications and security definitions. Still, all of them can be viewed as instances of the data authentication primitive. Therefore, our results (which apply to any data authentication primitive) apply to each of the standard primitives.

**Equivalence of Security Against One and Multiple Verification Queries for Data Authentication Primitives.** As one of our main results we show that for certain data authentication primitives security against one verification query implies security against multiple queries. This result holds for a class of data authentication primitives that have efficient revealing functions. Informally, a data authentication primitive has the revealing function if an adversary can efficiently compute answers to those verification queries which are not fresh without calling a verification oracle.

In Theorem 3.4.2 we prove that for any data authentication primitive which has a revealing function, security against a single verification query implies security against
multiple queries. As a corollary we obtain the equivalence result for MA schemes under strong unforgeability and for AE schemes under integrity of ciphertexts.

### 1.1.5 Repair: Recovering Security of Practical MA Schemes

The fact that MA-UF security against one query doesn’t imply security against multiple queries means that what has been proved about specific existing MA schemes does not guarantee their security under multiple forgery attempts in practice. However, it does not say that these particular schemes are actually insecure under multiple verification attempts. We now ask whether it may be possible, via other routes, to prove them secure against multiple verification queries.

In answering this question, it would be much preferable to avoid re-entering the (sometimes complex) proofs of security of these MA schemes. Our approach, instead, is to consider some special classes of MA schemes such that (1) known schemes have been proved to fall into one of these classes, and (2) we can show that any scheme in the class is MA-SUF secure. Then, via our equivalence implication for MA-SUF security, and existing results about the schemes, we can conclude that they are secure against multiple verification queries. We implement this program as follows.

The first class of MA schemes we consider are message authentication codes or MACs. (A MA scheme is a message authentication code, or MAC, if its tagging function is stateless and deterministic, and verification is done by applying the tagging function to compute the correct tag of the given message and comparing this with the candidate tag.) It is easy to see (cf. Proposition 3.6.1) that any MA-UF secure MAC is in fact MA-SUF secure, and hence (by Corollary 3.4.3) secure against multiple verification queries. As an application, since HMAC, proven MA-UF secure against one verification query in [3], is a MAC, the above tells us that it is both MA-UF and MA-SUF secure against multiple verification queries.

An important subclass of MACs are PRF based ones, where the tagging function is pseudorandom. These are MA-UF secure against one verification query [33, 10] and thus by the above automatically MA-SUF secure against multiple queries. Since the
tagging algorithms of block-cipher based constructs like various CBC-MACs [10, 19, 53], PMAC [18], TMAC [45] and OMAC [39] are shown in the cited papers to be PRFs, we can conclude that the corresponding MACs are $\text{MA-UF}$ and $\text{MA-SUF}$ secure against multiple queries.

The second class of ma-schemes we consider are Carter-Wegman (CW) MA schemes [62]. These schemes are randomized or counter-based, and hence not MACs, that is why Proposition 3.6.1 does not apply. However, via an extension of the standard proof establishing the $\text{MA-UF}$ security against one query of these ma-schemes [44, 57, 36, 16, 17], we can establish their $\text{MA-SUF}$ security against one query as well. Hence by Corollary 3.4.3, they too are $\text{MA-UF}$ and $\text{MA-SUF}$ secure against multiple queries. In particular, UMAC [16, 17] and MMH-MAC [36] are CW ma-schemes shown in the cited papers to be $\text{MA-UF}$ security against one query, and hence are $\text{MA-UF}$ and $\text{MA-SUF}$ secure against multiple queries by the above.

Finally, we establish security of the generalized Carter-Wegman MA schemes [7], whose security proof we found to be incomplete. We directly prove that generalized Carter-Wegman MA schemes are $\text{MA-SUF}$ secure against multiple verification queries and this security proof yields better concrete security than first proving security against one query and then applying Theorem 3.4.2.

1.1.6 Quality of the Equivalence Reduction

Quality of the Equivalence Reduction, and Its Impact. The reduction of Theorem 3.4.2, which establishes that security of data authentication primitives against one verification query implies that against multiple queries, is not tight: given a polynomial time adversary $A$ making $q_v$ verification attempts and succeeding with probability $a$, it delivers a polynomial time adversary $B$ making one verification attempt and succeeding with probability at least $b = a/q_v$. The above-mentioned results showing security of various cryptographic primitives inherit this loss in security through their reliance on Theorem 3.4.2. Unfortunately this loss can be significant in practice. Con-
sider for example PRF-based MACs using a block cipher with block-length $n$. Typically [10, 19, 53, 18, 45, 39] it is shown that the probability of forgery under $t$ tagging queries, each of at most $m$ blocks, is roughly $m^2 t^2 / 2^n$. With AES ($n = 128$) and assuming 1Kbyte messages ($m = 2^{13}$) this provides MA-SUF security against one verification query until around $t = 2^{51}$ messages are tagged, which is adequate if we only cared about one verification query, but of course we don’t. Obtaining MA-SUF security against multiple queries via Theorem 3.4.2, we get that the probability of forgery under $q_v$ verification queries is roughly $q_v m^2 t^2 / 2^n$, with $t$, $m$ as before. Now, for example, a guarantee of security against $q_v = 2^{34}$ verification attempts requires that at most $t = 2^{34}$ messages are tagged. These numbers are not large enough for comfort, and things get substantially worse if we consider legacy ciphers with block length $n = 64$.

**NO BETTER GENERAL REDUCTION.** The first question to ask given the above is whether there is a better reduction showing security of data authentication primitives against one verification query implies that against multiple queries. Proposition 3.5.1 shows that the answer is negative. It does this by presenting a data authentication primitive (which is also a MA scheme) for which there is a polynomial-time attack having constant probability of a forgery with $q_v$ queries, but for which no polynomial-time attack making only one verification query succeeds in a forgery with probability significantly better than $1/q_v$. This shows (cf. Corollary 3.5.2) that no reduction could deliver a reduction factor that is better than that of Theorem 3.4.2 by more than a constant factor, even for the special case of data authentication primitives that are MA schemes.

### 1.1.7 Concrete Security Improvements

**IMPROVEMENTS FOR DATA AUTHENTICATION PRIMITIVES.** Although a general concrete security improvement is ruled out by the above, we are able to find such improvements for important classes of data authentication primitives.

Many data authentication primitives have certain random coins for the security experiment, which trivially allow an adversary to do the forgery (for example, in MA
schemes with random nonces this is the event that two nonces collide). Denote the probability of this happening by \( \varepsilon_1 \) and let \( \varepsilon_2 \) be maximal adversary’s advantage against one verification query in the case that the unfortunate event doesn’t happen.

Theorem 3.4.2 applied to this data authentication primitive provides concrete security

\[ q_v (\varepsilon_1 + \varepsilon_2) \]

against adversaries making at most \( q_v \) verification queries. Theorem 3.4.5 allows to improve concrete security of data authentication primitives against multiple verification queries over what implied by Theorem 3.4.2 and provides concrete security

\[ \varepsilon_1 + q_v \cdot \varepsilon_2. \]

**Improvements for Particular MA Schemes.** We also improve concrete security for important subclasses of MA schemes. Proposition 3.6.4 presents an essentially tight reduction of MA-SUF security to the pseudorandomness of the MAC in the case the latter is PRF-based. Continuing the above example, this will imply that the probability of forgery under \( v \) verification queries is roughly \( m^2 (v^2 + t^2) / 2^n \), with \( v, t, m \) as before. Now (for 1Kbyte messages) we have a guarantee of security against \( v = 2^{50} \) verification attempts even if up to \( t = 2^{50} \) messages are tagged.

This is a substantial improvement. It is particularly valuable since so many ma-schemes are PRF-based [10, 19, 53, 18, 45, 39], and we note it can also be applied to HMAC, whose tagging algorithms is often assumed to be a PRF, due to its usage for tasks such as key-derivation. (We note the proof of [3] extends to establish its security as a PRF if we are willing to assume the underlying compression function is a PRF.

**Improvements for the EAX Mode of Operation.** We use the results on data authentication primitives to improve concrete security of the EAX mode of operation. EAX is a mode of operation of a block cipher which provides for authenticated encryption with associated data; it was introduced and analyzed by Bellare, Rogaway and Wagner [15]. EAX mode can be applied to any block cipher \( E \) and involves a parameter \( \tau \), which specifies the tag length. EAX mode takes a triple \((N, M, H)\) of strings from
\{0, 1\}^* \), where \( N \) is a nonce, \( M \) is a message and \( H \) is a header, and produces a ciphertext \( C \) of length \(|M| + \tau\). EAX mode can use any algorithm for computing nonces: the security experiment for AEAD schemes grants an adversary the power to select arbitrary nonces to be used by the encryption oracle with the only restriction for nonces being all distinct.

EAX mode was proved secure [57] if the block cipher is modeled by a random permutation. No adversary with data complexity \( \sigma \) (maximal total length of all encryption queries) who makes at most \( q_v \) verification queries can break authenticity of EAX mode with probability better than

\[
q_v \cdot \left( \frac{10.5\sigma^2}{2^n} + \frac{1}{2^\tau} \right).
\]

We observe that this reduction is loose in terms of concrete security it provides. For example, for 128-bit block ciphers the security proof of Bellare et al. [15] only provides security against adversaries running in time at most \( 2^{41} \), which is more than feasible for nowadays computational resources.

The authors of EAX [15] conjectured without a proof that concrete security of EAX is indeed better: specifically, they conjectured that it is possible to show that advantage of adversaries with data complexity \( \sigma \) who make at most \( q_v \) verification queries is bounded by

\[
\frac{10.5\sigma^2}{2^n} + \frac{q_v}{2^\tau}.
\]

Our results allow to fill this gap in the security analysis of EAX and establish the conjectured bound. The improved result applied to 128-bit block ciphers provides security against adversaries running in time at most \( 2^{62} \).

1.1.8 Remarks

One might consider a multiple verification version of the definition of digital signatures as well, but it is clearly equivalent to the standard definition of [35] because verification takes place under a key that is public (and in particular available to the adversary).
In practice, “throttling” is often used to limit the number of verification attempts an adversary can make. (The verification server refuses further requests under a key for which some number of verification attempts have been previously rejected). It would be a mistake to think that the use of throttling means that security against a single verification query suffices. For one thing, one surely cannot limit a user to just one verification attempt before revoking their key, for a few rejections can occur for natural reasons such as corrupted transmissions. Also, revocation of keys brings key-management costs. Thus we feel that for practice the most desirable situation is to have schemes providing security against a large number of verification attempts.

1.2 Append-Only Signatures

1.2.1 Definition

In many real-world applications, users and programs alike require notions of authentic delegation of data to model the flow of information. It is often required that delegation from one party to another enables the delegatee to “append” to the information it received but to do nothing more. For example, in wide-area Internet routing, each network passes a routing path advertisement to its neighboring networks, which then append to it information about themselves and forward the updated advertisement to their neighbors. For security, the route advertisements must be authenticated; intermediate networks must be incapable of modifying routes except according to the protocol (that is, by appending their names to already-received advertisements). Likewise, in the context of secure resource delegation for distributed systems, users need to delegate their share of resources to other users, who may then re-delegate to other users by including their own resources in the pool. In many of these applications, it is desirable that delegation is possible without parties having to share any cryptographic keys and that the authenticity of any information received through a series of delegations is verifiable based only on the identity of the first party in the chain.

To directly address the needs of these applications, we present a new cryptographic
primitive for data authentication called Append-Only Signatures (AOS). Informally, an AOS scheme enables the extension of signed messages and update of the corresponding signatures, without requiring possession of the signing key. That is, any party given an AOS signature $\text{Sig}[M_1]$ on message $M_1$ can compute $\text{Sig}[M_1 || M_2]$ for any message $M_2$, where $M_1 || M_2$ is the concatenation of $M_1$ and $M_2$. The verifier of the final signature needs the initial signer’s public key but does not need to know the public keys or any other information from intermediate signers except the message data appended. Clearly, such a scheme cannot be secure according to the standard notion of security for signatures. Instead, we define an AOS scheme to be secure if it is infeasible to forge signatures of messages that are not obtained by extending already-signed messages. We define the security of AOS more formally in Section 4.1.1.

### 1.2.2 Constructions

We present several provably secure AOS schemes, offering different tradeoffs of flexibility and efficiency. Our first construction shows a generic approach to building AOS schemes from any standard digital signature scheme $\text{SIG}$ using certificate chains. The construction works as follows: The secret and public keys for the AOS scheme are obtained by running the key generator for $\text{SIG}$. For any message $M = M_1 || M_2 || \cdots || M_n$, each $M_i$ being a symbol in some predetermined message space, the AOS signature of $M$ is defined as a sequence of $n$ public keys $pk_1, pk_2, \cdots, pk_n$ (generated using the key generator for $\text{SIG}$) and a sequence of $n$ certificates binding the message symbols to these public keys. The $i$th certificate in the chain binds the message symbol $M_i$ to the corresponding public key $pk_i$ and is signed using the secret key, $sk_{i-1}$, corresponding to $pk_{i-1}$. The secret key, $sk_0$, of the AOS scheme signs the first certificate in the chain while the secret key $sk_n$ (corresponding to the last public key), is revealed as part of the AOS signature and is used for appending new symbols to $M$. We observe that if the message space is small enough, we can make use of “weaker”, and more efficient, signature schemes without compromising the security of the resulting AOS scheme. Furthermore, using aggregation techniques[24, 46], the size of the certificate
chain can be made a constant, that is, independent of the number of message symbols appended, which leads to shorter AOS signatures than those in the basic scheme. (See Section 4.2 for more details on this scheme.)

Since signature schemes exist given the existence of one-way functions [58], the above construction implies the existence of AOS under the same assumption. We also present a more efficient construction of AOS for applications in which the message space is constant size and the total number of append operations performed is also constant. This construction is based on a stronger assumption and makes use of pseudorandom generators and collision-resistant hash functions (CRHFs). We remark that both these schemes—one using CRHFs and one based on certificate chains—are in the standard model; neither of them makes use of random oracles.

1.2.3 Relation to Hierarchical Identity-Based Signatures

Identity-Based Signature (IBS) schemes, due to Shamir [59], are signature schemes in which the identity of the signer (for example, her email address) plays the role of his public key. Such schemes assume the existence of a trusted authority that holds a master public-private key pair that is used to assign secret keys to users based on their identities. Anyone can verify signatures on messages signed by a user knowing only the master public key and the identity of that user. Hierarchical IBS (HIBS) schemes, proposed by Gentry and Silverberg [29], are a natural generalization of this concept to a setting in which users are arranged in a hierarchy and a user at any level in this hierarchy can delegate secret keys to her descendants based on their identities and her own secret key. To verify the signature created by any user, one needs to know the identity of the user (and her position in the hierarchy) and the public key of the root user.

HIBS can be implemented using certificate chains (as suggested in [29]) and the resulting construction bears a strong resemblance to the certificate-based construction of AOS we provide. Upon closer examination, we find that the similarity between the two constructions is not accidental: it is an artifact of the close relationship between the two primitives themselves—AOS and HIBS are, in fact, tightly equivalent. This means that
(a) there exist generic transformations from any HIBS scheme into a corresponding AOS scheme and, likewise, from any AOS scheme into a corresponding HIBS scheme; and (b) these transformations are extremely efficient (the derived scheme is as efficient as the scheme being derived from) and highly security-preserving (an adversary attacking the derived scheme can be transformed into an adversary attacking the original one, losing only a constant factor in efficiency and query complexity).

A benefit of this equivalence is that it considerably simplifies the notion of HIBS and makes security analysis for HIBS schemes less onerous: AOS is simpler than HIBS, and, for any HIBS scheme, it is typically easy to find an equivalent AOS scheme whose security properties carry over to the corresponding HIBS scheme. For example, our security proof for certificate-based AOS translates to a security proof for certificate-based HIBS (originally proposed in [29]). Although this construction of HIBS was known prior to our work, it was never analyzed in the literature, and, to the best of our knowledge, we give the first proof of security for it. Furthermore, our construction of AOS based on pseudorandom generators and CRHFs yields a novel approach to designing HIBS and can be useful for some restricted scenarios (for example, in a constant-depth hierarchy wherein each user signs messages from a constant-size message space). We remark that both these constructions yield HIBS schemes in the standard model and neither involves the use of computationally intensive bilinear maps (this is in contrast with some recent results on HIBS [26]). Finally, we believe that AOS is a more intuitive primitive to study because it clearly reflects the requirements of our problem domains (such as secure routing) and is better suited for analyzing the problems that motivate our work.

1.2.4 Related Work

Append-only signatures belong to a general class of primitives called algebraic signatures. Informally, an algebraic signature scheme allows the creation of signatures on a new message $M$ using the signatures on some known messages, $M_1, M_2, \ldots, M_n$, and the public key, provided the new message can be obtained from the known messages
using some prespecified set of \((n\text{-ary})\) operations, say \(O = \{f_1, f_2, \cdots, f_m\}\). That is, given the signatures, \(\text{sig}[M_1], \ldots, \text{sig}[M_n]\) and the public key, it is easy to compute \(\text{sig}[f_i(M_1, \ldots, M_n)]\) for any \(f_i \in O\). In our setting, each \(f_i\) has arity 1 and appends some fixed message symbol \(M_i\) to an input message \(M\). Security for algebraic signatures is defined in a manner similar to our approach to security of AOS (that is, it should be hard to forge signatures of messages that cannot be obtained by applying the operations in \(O\) to already-signed messages). Examples of algebraic signatures studied in the literature include transitive signatures by Micali and Rivest [49], homomorphic signatures by Johnson, Molnar, Song and Wagner [41], and graph-based algebraic signatures by Hevia and Micciancio [37].

Although no obvious relation exists between our primitive and any of the previously studied algebraic signature primitives, we do note that some of the techniques we use in our constructions parallel prior techniques. For example, our construction of AOS schemes using CRHFs can be viewed as a special instance of graph-based algebraic signature schemes studied in [37] (although the set of update operations considered there are different from the \text{append} operation that we consider). Also, [41] introduces the notion of \textit{redactable signatures}—signatures that allow “deletion” of message symbols without access to the secret key—and one of our constructions for this primitive is also an example of graph-based algebraic signatures.

A concept closely related to AOS (and algebraic signatures, in general) is that of \textit{incremental signatures}, proposed by Bellare, Goldreich, and Goldwasser [5, 6]. Given an incremental signature on a message \(M\), it is possible to compute the signature on a slightly updated version of \(M\) in time proportional to the “amount” of change made to \(M\) (rather than on the length of the entire message). The update operation, however, requires access to the initial signer’s secret key whereas, in the case of AOS, a message can be updated by any party. Moreover, the update operations considered in [5, 6] are \textit{replace}, \textit{insert} and \textit{delete} whereas we are interested in performing \textit{append} operations on messages.
1.2.5 Application to Secure Routing

In our discussion of applications of AOS, which we expand upon in Section 4.5, our main focus is on the problem of wide-area Internet routing security. We argue that the existence of secure AOS schemes is a sufficient condition for guaranteeing an important security property of routing protocols, namely, authenticity of route announcements. Though routing is an important component of modern communication networks, its security has received little attention from the cryptography community (although a rich literature exists in the computer networks community [43, 60, 38, 61]). Most cryptographic protocols assume that the network is an untrusted black box possibly under full control of the adversary; while this is useful when modeling the security of end-to-end protocols, it fails to capture the security of the underlying routing protocols. We are motivated by this apparent gap, as routing is not just an application in which cryptography is required, but a necessary component of the networks used by most cryptographic protocols.
2 Security of MA and AE Schemes against Multiple Verification Queries

2.1 Definitions

NOTATION. If $A$ is a randomized algorithm then $a \leftarrow A(x, y, \cdots)$ means that $A$ is executed on inputs $x, y, \ldots$ with fresh coins and $a$ denotes the outcome. If $S$ is a set then $s \leftarrow S$ means that $s$ is chosen uniformly at random from $S$. If $x, y$ are objects then $x \leftarrow y$ means $x$ is assigned the value $y$.

MESSAGE AUTHENTICATION SCHEMES. A message authentication scheme (MA scheme) $\Pi = (\text{SGN}, \text{VF})$ over sets $K, M$ and $T$ is a pair of polynomial-time algorithms, where

- The tag-generation or signing algorithm $\text{SGN}$, which may be randomized or stateful, takes a key $K \in K$ and a message $M \in M$ to return a tag $T \in T$.

- The deterministic, stateless verification algorithm $\text{VF}$ takes a key $K \in K$, a message $M \in M$ and a candidate tag $T \in T$ to return either 1 (ACCEPT) or 0 (REJECT).
Intuitively, $K$, $M$ and $T$ represent sets of keys, messages and authentication tags respectively. We require the following completeness condition: $\text{VF}(K, M, \text{SGN}(K, M)) = 1$ with probability 1 for any $K \in K$ and any $M \in M$. We say that $\Pi$ is a message authentication code (MAC) if SGN is stateless and deterministic, and also $\text{VF}(K, M, T)$ is defined via: If $T = \text{SGN}(K, M)$ then return 1 else return 0.

Adversaries against MA Schemes. An adversary against a MA scheme $\Pi$ over sets $K$, $M$ and $T$ is a probabilistic algorithm with access to signing oracle $\text{sig}(\cdot)$ and verification oracle $\text{vf}(\cdot, \cdot)$. The former oracle takes input messages from $M$ and returns tags from $T$ and the later takes input tuples from $M \times T$ and returns either 0 or 1.

We find it useful to quantify resources available to adversaries. We say that adversary $A$ is a $(t, q_s, q_v)$--adversary if $A$ runs in time at most $t$, makes at most $q_s$ signing queries and at most $q_v$ verification queries. We set either of $t, q_s$ or $q_v$ to $\infty$ if the adversary is unbounded in a use of that resource. We will call adversaries who make at most one verification query as single-query adversaries and those making multiple verification queries as multiple-queries adversaries.

Security Games. Throughout this work we will define security of various cryptographic primitives via security games. A typical security game $\text{TYPE}$ is played with a party $A$ called the adversary; we denote the execution of game $\text{TYPE}$ with $A$ by $A^{\text{TYPE}}$. All security games will involve a boolean variable $\text{win}$, which is initialized with false in the beginning of the game. We say that $A^{\text{TYPE}}$ sets $\text{win}$ or that $A$ wins the game $\text{TYPE}$ if at some point in the game variable $\text{win}$ is set to true.

Two Security Notions. We provide the formal definitions and then some explanations. Let $\Pi = (\text{SGN}, \text{VF})$ be a message authentication scheme. Let $\text{TYPE} \in \{\text{MA-UF}, \text{MA-SUF}\}$; MA-UF denotes a notion of standard unforgeability of MA schemes and MA-SUF denotes strong unforgeability. Let $A$ be any adversary against $\Pi$. We define security game $\text{TYPE}$ as depicted in Figure 2.1. We let

$$\text{Adv}^{\text{TYPE}}_\Pi (A) = \Pr \left[ A^{\text{TYPE}} \text{ sets } \text{win} \right]$$
Game $\text{TYPE}$

\[ K \overset{\$}{\leftarrow} K \; ; \; \text{win} \leftarrow \text{false} \]

- When $\mathcal{A}$ makes a query $M$ to $\text{sig}(\cdot)$ do

\[ T \overset{\$}{\leftarrow} \text{SGN}(K, M) \; ; \; \text{TagSet}[M] \leftarrow \text{TagSet}[M] \cup \{ T \} \; ; \; \text{Return T to A} \]

- When $\mathcal{A}$ makes a query $(M, T)$ to $\text{vf}(\cdot, \cdot)$ do

\[ d \leftarrow \text{VF}(K, M, T) \]

If $d = 1$ then

- If (TYPE = MA-UF and TagSet[$M$] = $\emptyset$) then \( \text{win} \leftarrow \text{true} \)

- If (TYPE = MA-SUF and $T \not\in \text{TagSet}[M]$) then \( \text{win} \leftarrow \text{true} \)

\[ \text{Return d to A} \]

Figure 2.1 Security game defining standard (MA-UF) and strong (MA-SUF) security of MA scheme $\Pi = (\text{SGN, VF})$ with key space $K$.

denote the winning probability for $\mathcal{A}$ in the security game in question. We say that $\Pi$ is $\text{TYPE}$ $\varepsilon$–secure against $(t, q_s, q_v)$–adversaries if for any such adversary $\mathcal{A}$,

\[ \text{Adv}^\text{TYPE}_{\Pi}(\mathcal{A}) < \varepsilon. \]

We say that $\Pi$ is $\text{TYPE}$ secure (against single or multiple verification queries) if concrete security provided by $\Pi$ and a class of adversaries are clear from the context.

In the security game $\mathcal{A}$’s signing oracle queries are answered via $\text{SGN}(K, \cdot)$ and its verification oracle queries via $\text{VF}(K, \cdot, \cdot)$. The rest is book-keeping. Depending on $\text{TYPE}$, $\text{win}$ is set to $\text{true}$ if the conditions of a standard forgery or a strong forgery are met. It is assumed that the set $\text{TagSet}[M]$ is initially empty for all $M \in M$.

PRFs and PRF-based MACs. Let $F$ be a function with domain $K \times M$ and range $T$. Recall [33, 10] that if $B$ is an adversary with an oracle for a function with domain $M$ and range $T$, then its PRF-advantage is

\[ \text{Adv}^\text{PRF}_F(B) = \Pr[B^{F(K, \cdot)} = 1] - \Pr[B^{f(\cdot)} = 1], \]
the first probability being over $K \mathrel{\triangleq} K$ and the second over $f \mathrel{\triangleq} \text{Maps}(M, T)$, where Maps$(M, T)$ is the set of all functions mapping from domain $M$ to range $T$.

We associate to $F$ the MA scheme $\Pi[F] = (F, VF)$ over sets $K, M$ and $T$, where $VF(K, M, T)$ returns 1 if $F(K, M) = T$ and 0 otherwise. A MA scheme $\Pi$ is said to be PRF-based if there is a function $F$ such $\Pi = \Pi[F]$. Note that a PRF-based MA scheme is a MAC.

### 2.2 Template of a Reduction

We investigate how security of MA schemes depends on the number of verification oracle queries the adversary is allowed to make. In practice, it is certainly possible for an adversary to make multiple verification attempts. (For example a server authenticating a client under their common key functions as the verification oracle, and an adversary can forward many transmissions to it.) A scheme that is secure against a single verification query but not secure against multiple verification queries is thus clearly not providing adequate security for practical usage.

It turns out that this fact (namely, that security against multiple verification queries is the “right” model) is actually quite well understood in the community and literature. Nonetheless, it is the MA-UF notion under a single verification query that most schemes are proven to meet [10, 53, 3, 19, 18, 16, 17, 45, 39, 40]. (Exceptions are [8, 7].)

It appears to be due to the belief that MA-UF security under a single verification query implies MA-UF security under multiple queries. More precisely, the belief is that if an adversary $A$ makes $q_v$ verification oracle queries, its advantage (probability of forgery) is not more than $q_v$ times that of an adversary $B$ of comparable time making just one verification oracle query. So the belief is that although a difference in concrete security does manifest itself, securities against single-query and multiple-queries adversaries are polynomially equivalent. This view is expressed for example in [17, Page 21], who say:

This definitional choice is pretty inconsequential ... generalizing to $q_v \geq 1$ verification oracle queries will increase the adversary’s chance of success.
by at most $q_v$. The proof is simple and the observation is well-known, so the proof is omitted.

The “choice” they refer to is whether to allow one or many queries to the verification oracle. The “proof” they refer to does not appear anywhere, but the folklore argument is that $B$ can guess one of the verification queries of $A$ to use as its own, single verification query, answering previous ones by “0.”

Let $\Pi = (\text{SGN}, \text{VF})$ be a MA-UF secure MA scheme against single-query adversaries. As we noted earlier, there is a belief that one can prove it also secure against multiple-queries adversaries. Here we will present the template of the “proof” that people appear to have in mind and see why it does not work. This is useful for two reasons: it lends some insight into the later counter-example showing that this implication simply isn’t true, and the similar idea will be later appropriately used to show that this implication holds for certain other data authentication primitives.

Let $A$ be a $q_v$-query adversary attacking $\Pi$. We want to define a single-query adversary $B$ such that the following inequality, which we refer to as Eq$[A, B]$ holds:

$$\text{Eq}[A, B] : \quad \text{Adv}_{\Pi}^{\text{MA-UF}}(B) \geq \frac{1}{q_v} \cdot \text{Adv}_{\Pi}^{\text{MA-UF}}(A).$$

The idea is for $B$ to guess and output the first successful verification query made by $A$, answering all others negatively without consulting its verification oracle. The corresponding adversary, which we call $B_1$, is defined by the part of the code of Figure 2.2 that omits the boxed statements. (Ignore the boxed statements for now.) Let $I$ be the random variable, in the execution of $A^{\text{MA-UF}}$, that takes as value the first verification query made by $A$ to which the answer of the verification oracle is 1 and the message was not previously queried to the tagging oracle. Now we would like to claim that the guess made by $B_1$ equals $I$ with probability at least $1/q_v$, and thus Eq$[A, B_1]$ is true.

However, this is not true. For example, $A$ might make a query $M$ to its tagging oracle, get back a tag $T$, and immediately make query $(M, T)$ to its verification oracle. Let us imagine that, after this, it makes another verification oracle query $(M', T')$ that is valid, with $M' \neq M$, so that it wins with probability one. However, $B_1$’s simulation is
Adversary $B^{\operatorname{sig}(:,.)}_{b} (b \in \{1, 2\})$

\begin{align*}
\text{count} &\leftarrow 0; \text{guess} \leftarrow \{1, \ldots, q_v\} \\
\text{Run } A^{\operatorname{sig}(:,.), \operatorname{vf}(:,.)} \text{ and reply to its oracle queries as follows:}
\end{align*}

- When $A$ makes a query $M$ to $\operatorname{sig}(:,.)$ do
  \begin{align*}
  &\text{Obtain } T \leftarrow \operatorname{sig}(M) \\
  &\text{TAGSET}[M] \leftarrow \text{TAGSET}[M] \cup \{T\} \\
  &\text{Return } T \text{ to } A
  \end{align*}

- When $A$ makes a query $M, T$ to $\operatorname{vf}(:,.)$ do
  \begin{align*}
  &\text{count} \leftarrow \text{count} + 1 \\
  &\text{If (count} > \text{guess) then Halt} \\
  &\text{If (T} \in \text{TAGSET}[M]) \text{ then return 1 to } A \\
  &\text{If (count} < \text{guess) then return 0 to } A \\
  &\text{If (count} = \text{guess) then obtain } d \leftarrow \operatorname{vf}(M, T) ; \text{ Halt}
  \end{align*}

Figure 2.2 Adversaries $B_1, B_2$ derived from the adversary $A$. Adversary $B_1$ omits the boxed statements while $B_2$ includes them.

inaccurate, because (in the case guess $\geq 2$) the verification oracle would have returned 1 in answer to $A$’s first query to it, but $B_1$ returns 0 to $A$ as the answer. So $B_1$’s advantage is 0. (In the case guess $= 1$, $B_1$ does not win because $M$ was queried to the tagging oracle and is not new).

This seems easily fixed by comparing tags in verification oracle queries to ones returned previously by the tagging oracle. Namely, we consider adversary $B_2$ of Figure 2.2 that now includes the boxed statements. For our example above, $B_2$’s simulation is now correct, and thus $\text{\LTL{Eq}[A, B]}$ is true. However, in general, this strategy is still wrong, meaning there are adversaries $A$ for which $\text{\LTL{Eq}[A, B_2]}$ does not hold. The reason is malleability. Suppose $A$ begins by making a query $M$ to its tagging oracle, getting back a tag $T$, and is then capable of modifying $T$ to some different value $\overline{T}$ which is valid tag for $M$. Let it then make query $(M, \overline{T})$ to its verification oracle. As before, imag-
ine that, after this, it makes another verification oracle query \((M', T')\) that is valid, with \(M' \neq M\), so that it wins with probability one. Now, again, \(B_2\)’s simulation is inaccurate, because (in the case \(\text{guess} \geq 2\)) the verification oracle would have returned 1 in answer to \(\mathcal{A}\)’s first query to it, but \(B_2\) returns 0 to \(\mathcal{A}\) as the answer. So \(B_2\)’s advantage is 0.

This time, it is not clear how to fix \(B_2\), because in general it is not clear how to detect whether \(T\) is a valid tag for \(M\) without querying the oracle. Theorem 2.3.1 implies that this difficulty is not surmountable.

### 2.3 MA-UF Security against Single Verification Query
doesn’t Imply Security against Multiple Queries

The above discussion leaves open the question of whether MA-UF security against single-query adversaries implies MA-UF security against multiple-queries adversaries. Perhaps the proof could be patched? The following theorem implies that it cannot, because the underlying claim is simply not true.

**Theorem 2.3.1** Assume there exists a MA scheme which is MA-UF \(\varepsilon\)-secure against \((t, q_s, 1)\)-adversaries. Then there exists a MA scheme, which is MA-UF \(\varepsilon\)-secure against \((t', q_s, 1)\)-adversaries (for \(t' = t - O(q_s)\)) but which can be broken with probability 1 by some efficient multiple-queries adversary. □

The rest of this section is devoted to proving Theorem 2.3.1. By assumption, there exists a MA-UF secure MA scheme \(\Pi = (\text{SGN}, \text{VF})\) over sets \(K\), \(M\) and \(T\). Without loss of generality we can assume that \(K \subset \{0, 1\}^l\) for some \(l > 0\) and thus for any key \(K \in K\) we can assume that \(K \in \{0, 1\}^l\). For \(i \in \{0, \ldots, l\}\) we denote by \(K[i]\) the \(i\)-th bit of \(K\). From \(\Pi\) we build another MA scheme \(\overline{\Pi} = (\overline{\text{SGN}}, \overline{\text{VF}})\) over sets \(K\), \(M\) and \(T \times \{0, \ldots, l\}\) whose constituent algorithms are specified in Figure 2.3.

We first note that the scheme satisfies the completeness condition. Indeed, if \(\overline{T}\) is an output of \(\overline{\text{SGN}}(K, M)\) then it has the form \((T, i)\) with \(i = 0\). Thus the first “If” statement of \(\overline{\text{VF}}(K, M, T)\) applies and returns \(d\), which is 1 by completeness of \(\Pi\). Now
Algorithm $\text{SGN}(K, M)$

\[
T \leftarrow \text{SGN}(K, M) \\
\overline{T} \leftarrow (T, 0) \\
\text{Return } \overline{T}
\]

Algorithm $\text{VF}(K, M, \overline{T})$

Parse $\overline{T}$ as $(T, i)$ where \( i \in \{0, 1, \hdots, l\} \)

\[
d \leftarrow \text{VF}(K, M, T) \\
\text{If } (d = 0 \text{ or } i = 0) \text{ then return } d \\
\text{If } (d = 1 \text{ and } i \geq 1) \text{ then return } K[i]
\]

Figure 2.3 Message authentication scheme $\Pi$ used in the proof of Theorem 2.3.1.

we present an attack to show that $\Pi$ is not secure against an adversary making more than one query to its verification oracle.

**Claim 2.3.2** $\Pi$ is not MA-U$\text{F}$ secure against multiple-queries adversaries: there exists $(O(l), 1, l+1)$–adversary $\mathcal{A}$ who succeeds with probability 1.

**Proof:** Consider the following MA-U$\text{F}$ adversary $\mathcal{A}$ against $\Pi$:

Adversary $\mathcal{A}^{\text{sig}(\cdot), \text{vf}(\cdot, \cdot)}$

Pick any message $M_1 \in \mathcal{M}$ and obtain $T_1 \leftarrow \text{sig}(M_1)$

Parse $T_1$ as $(T_1, 0)$

For $i = 1, \hdots, l$ obtain $L[i] \leftarrow \text{vf}(M_1, (T_1, i))$

Pick any message $M_2 \in \mathcal{M} \setminus \{M_1\}$

$T_2 \leftarrow \text{SGN}(L, M_2)$; obtain $d \leftarrow \text{vf}(M_2, T_2)$

By examining the construction of $\mathcal{A}$, we find that $\mathcal{A}$ runs in time $O(l)$, makes 1 signing query and $l + 1$ verification queries.

Adversary $\mathcal{A}$ first obtains the tag of a message $M_1$ using its tagging oracle. The definition of the verification algorithm for $\Pi$ then tells us that, above, $L[i] = K[i]$ is the $i$-th bit of the key $K$ for all $i \in \{1, \hdots, l\}$. In other words, $\mathcal{A}$ has succeeded in recovering the key. Then it can, of course, easily win, by forging the tag of some new message $M_2$. (Note that its computation of $T_2$ is not an oracle query. $\mathcal{A}$ simply runs algorithm $\text{SGN}$ with key $L$ and message $M_2$.) Thus $\text{Adv}_{\Pi}^{\text{MA-U$\text{F}$}}(\mathcal{A}) = 1$.

Next we show that $\Pi$ retains the MA-U$\text{F}$ security of $\Pi$ against single-query adversaries. The intuition is simple. In order to make $\text{VF}(K, \cdot, \cdot)$ accept, an adversary must have a
message-tag pair that is valid for \( \text{VF}(K, \cdot, \cdot) \). On the other hand, being limited to one verification oracle query, it cannot make any use of any information that the verification oracle returns in answer to this query, since its game is effectively over once the query is made. Here now is the formal claim and proof.

**Claim 2.3.3** If \( \Pi \) is MA-UF \( \varepsilon \)-secure against \( (t, q_s, 1) \)-adversaries, then \( \Pi' \) is MA-UF \( \varepsilon \)-secure against \( (t', q_s, 1) \)-adversaries, where \( t' = t - O(q_s) \).

**Proof:** Let \( \mathcal{A} \) be any MA-UF \( (t', q_s, 1) \)-adversary attacking \( \Pi' \). We can assume it makes exactly one query to its verification oracle. We construct MA-UF \( (t, q_s, 1) \)-adversary \( \mathcal{A} \) attacking \( \Pi \) such that

\[
\text{Adv}_{\Pi}^{\text{MA-UF}}(\mathcal{A}) \leq \text{Adv}_{\Pi}^{\text{MA-UF}}(\mathcal{A}) \leq \varepsilon. \tag{2.1}
\]

Here is how \( \mathcal{A} \) works:

Adversary \( \mathcal{A}^{\text{sig}(\cdot), \text{vf}(\cdot, \cdot)} \)

- Run \( \mathcal{A} \) and reply to its oracle queries as follows:
  - When \( \mathcal{A} \) makes a query \( M \) to the signing oracle do
    - Obtain \( T \leftarrow \text{sig}(M) \); \( \overline{T} \leftarrow (T, 0) \)
    - Return \( \overline{T} \) to \( \mathcal{A} \)
  - When \( \mathcal{A} \) makes a query \( (M, \overline{T}) \) to the verification oracle do
    - Parse \( \overline{T} \) as \( (T, i) \) where \( i \in \{0, 1, \ldots, k\} \)
    - Obtain \( d \leftarrow \text{vf}(K, M, T) \)
    - Halt

\( \mathcal{A} \) does not return to \( \mathcal{A} \) any answer to \( \mathcal{A} \)'s (unique) verification-oracle query. (Not knowing the key \( K \), it would not know how). It simply uses this query to make its own verification-oracle query, and halts. Equation (2.1) is true because \( \mathcal{A} \)'s simulation of replies to queries of the tagging oracle is perfect, and also because \( \overline{\text{VF}}(K, M, (T, i)) \) returns 1 only if \( \text{VF}(K, M, T) \) returns 1.

This concludes the proof of Theorem 2.3.1. We now make some remarks that help relate this result to upcoming ones.
Remark 2.3.4 The MA scheme $\Pi$ of our counter-example is not a MAC. Although $\text{SGN}(\cdot, \cdot)$ is deterministic, the second requirement of MACs is violated: verification $\text{VF}(\cdot, \cdot, \cdot)$ is not done by recomputing $\text{SGN}(\cdot, \cdot)$ and comparing it with an input tag. This is relevant because otherwise we could contradict Proposition 3.6.1.

Remark 2.3.5 The MA scheme $\Pi$ of our counter-example is not MA-SUF secure against 1 verification query due to the following adversary:

Adversary $A^{\text{sig}(\cdot, \cdot), \text{vf}(\cdot, \cdot)}$

Pick any message $M \in M$ and obtain $T \leftarrow \text{sig}(M)$

Parse $T$ as $(T, 0)$ and obtain $d \leftarrow \text{vf}(M, (T, 1))$

The verification oracle accepts if $K[1] = 1$. Since $K$ is chosen at random we have $\text{Adv}_{\Pi}^{\text{MA-SUF}}(A) = 1/2$. This is relevant because otherwise we would reach a contradiction with Corollary 3.4.3.

2.4 On the Order of Signing and Verification Queries

We take a closer look at the security proofs for XOR MAC [8] and generalized Carter-Wegman MAC [7]. Both protocols were shown to be MA-SUF secure MA schemes against multiple-queries adversaries and in both cases security proofs make the same claim about the order in which adversaries make signing and verification queries. They claim that in order to prove MA-SUF security of their protocols it suffices to establish security against adversaries who first make all signing queries and then make all verification queries.

In the case of XOR MAC, a preliminary version of the paper [8] suggested without proof that verification queries could be postponed in their scheme, and the proof of the result given later in the full version [9] did not make this claim but used a different approach. In the case of generalized Carter-Wegman MAC [7], authors first proved that their scheme is secure if an adversary makes all signing queries prior to making any verification queries and then provided a (very brief) sketch on why this implies security
against general adversaries. We weren’t able to complete their sketch and moreover, the best security proof for their scheme we were able to find requires a different approach.

We do not know whether the original statements were true or not, but it motivates us to ask a general question, which we formulate in the following claim.

**Claim 2.4.1** Without loss of generality one can assume that an adversary against strong (MA-SUF) security of a MA scheme makes its verification queries only after it made all the signing queries. That is, for any MA scheme $\Pi$ and for any MA-SUF $(t, q_s, q_v)$–adversary $A$ against $\Pi$ one can construct $(O(t), q_s, q_v)$–adversary $B$ who makes all its signing queries prior to making any verification queries such that

$$\text{Adv}_{\Pi}^{\text{MA-SUF}}(B) \leq \text{Adv}_{\Pi}^{\text{MA-SUF}}(A).$$

Further on, we will call adversaries who can interleave signing and verification queries as interleaving adversaries and adversaries who make all verification queries after all signing queries as consecutive adversaries. We observe that Claim 2.4.1 in general is false. We construct a MA scheme, which provides strong unforgeability against consecutive adversaries and which can be broken by efficient interleaving adversaries.

**Proposition 2.4.2** For any integers $m$ and $n$ there exists a MA scheme $\Pi$ and an interleaving $(O(m), m, m)$–adversary $A$ such that

$$\text{Adv}_{\Pi}^{\text{SUF}}(A) = 1,$$

however for any consecutive $(\infty, \infty, q_v)$–adversary $B$ against $\Pi$

$$\text{Adv}_{\Pi}^{\text{SUF}}(B) \leq \frac{1}{m} + \frac{2q_v}{2^n}.$$ 

**Proof:** Consider the following MA scheme $\Pi = (\text{SGN}, \text{VF})$ with message space $\{0, 1\}^n$ and tag space $\{0, 1\}^n$. A key for $\Pi$ consists of a random function $F : \{0, 1\}^n \rightarrow \{0, 1\}^n$
Adversary $\mathcal{A}^{\text{sig}(\cdot), \text{vf}(\cdot)}$:

Set $x_0 \leftarrow 0^n$

For each $i = 0, \ldots, m - 1$

Query $(x_i, 0^n)$ to $\text{vf}(\cdot, \cdot)$

Query $x_i$ to $\text{sig}(\cdot)$ to obtain $x_{i+1}$

Figure 2.4 Construction of the adversary $\mathcal{A}$.

and of a random integer $t \in \{0, \ldots, m - 1\}$. For a fixed function $F$ we define $x_0^F$ to be $0^n \in \{0, 1\}^n$ and for $k = 1, \ldots, m - 1$ we recursively define $x_k^F = F(x_{k-1}^F)$. $\Pi$ consists of the following functions $\text{SGN}$ and $\text{VF}$:

$$\text{SGN}_{F,t}(M) = \begin{cases} 0^n, & \text{if } M = x_t^F; \\ F(M), & \text{otherwise.} \end{cases}$$

$$\text{VF}_{F,t}(M, T) = \begin{cases} 1, & \text{if } \text{SGN}_{F,t}(M) = T; \\ 0, & \text{otherwise.} \end{cases}$$

Intuitively, we consider a random function $F$ which is modified in a single point $x_t^F$ (this point is the $t$-th iteration of $F$ on $0^n$) by setting $F(x_t^F) = 0^n$. Signing algorithm $\text{SGN}$ consists of invoking modified function $F$ on a message while verification algorithm $\text{VF}$ re-computes $F$ on a message and compares the result with an input tag. If the random function $F$ was not modified, $(\text{SGN}, \text{VF})$ would be a perfectly secure MA scheme. As we show next, our modification makes the scheme less secure and the only way to break the scheme is to forge tag on a message $x_t^F$.

Consider the adversary $\mathcal{A}$ depicted in Figure 2.4. Clearly, $\mathcal{A}$ is an $(O(m), m, m)$–adversary. $\mathcal{A}$ wins MA-SUF security game with probability 1. Consider the execution of $\mathcal{A}$ and let $(F, t)$ be the secret key used in the forging experiment. We observe that for each $i \in \{0, \ldots, m - 1\}$, $x_i = x_t^F$. Therefore, at the first step $i$, when $x_i = x_t^F$ (this is guaranteed to happen for $i = t$ but also can accidentally happen for some $i < t$), the verification query made by $\mathcal{A}$ will be fresh and will be accepted by $\text{VF}$.

Now we are left to show that no consecutive adversary can win the MA-SUF game against $\Pi$ with high probability. Consider the execution of any consecutive $(\infty, \infty, q_v)$–
adversary \( B \), who first makes \( q_s \) signing queries \( M_1, \ldots, M_{q_s} \), receives the corresponding tags \( T_1, \ldots, T_{q_s} \) and then makes \( q_v \) verification queries \((M_1', T_1'), \ldots, (M_{q_v}', T_{q_v}')\).

First, since \( F \) is a random function, any fresh verification query \((M_i', T_i')\) such that \( M_i' \neq x_i^F \) will be accepted by the verification oracle with probability \( 1/2^n \). Therefore, the only non-negligible chance for \( B \) to win the experiment is to make a verification query \((x_i^F, 0^n)\). Otherwise, the winning probability of \( B \) is at most \( q_v/2^n \).

Next, we argue that \( x_i^F \) is either queried to the signing oracle or otherwise is hard to compute. Consider the values \( x_0^F, \ldots, x_{m-1}^F \). Denote by \( k \) the maximal integer such that \( \{x_0^F, \ldots, x_k^F\} \subset \{M_1, \ldots, M_{q_s}\} \). With probability \( k/m \), \( t \leq k \) and therefore \( x_i^F \in \{M_1, \ldots, M_{q_s}\} \) and \((x_i^F, 0^n)\) is known-authentic; otherwise (with probability \((m - k)/m \)) \( t > k \) and \( x_i^F \notin \{M_1, \ldots, M_{q_s}\} \)—in this case \((x_i^F, 0^n)\) would be a fresh query.

We claim that if \( t > k \), the probability that \( B \) guesses \( x_i^F \) (i.e. that \( x_i^F \in \{M_1', \ldots, M_{q_v}'\} \)) is at most \( 1/m + q_v/2^n \). With probability \( 1/m \), \( t = k + 1 \) and \( x_i^F \) is known to \( B \) as a tag returned by SGN on the signing query \( x_k^F \). Otherwise, \( x_i^F \) is a random point in \( \{0, 1\}^n \) and \( B \) (information-theoretically) cannot guess it any better than at random.

Summing up, we have that for any consecutive \((\infty, \infty, q_v)\)-adversary \( B \), the advantage of \( B \) in breaking strong unforgeability of \( \Pi \) is at most \( 1/m + 2q_v/2^n \).

### 2.5 Authenticated Encryption (INT-PTXT): Security against Single Verification Query Doesn’t Imply Security against Multiple Queries

#### 2.5.1 Definitions

An authenticated encryption scheme is a symmetric encryption scheme that simultaneously provides privacy and authenticity. We are not concerned here with the privacy.
Let us state the definitions of authenticity we consider.

**Two Security Notions.** A symmetric encryption scheme \( \Pi = (E, D) \) with key space \( K \), message space \( M \) and ciphertext space \( C \) is specified as usual via its encryption and decryption algorithms. The latter either returns a valid message or the symbol \( \perp \) to indicate rejection. We associate to the scheme a *verification algorithm* \( D^* \) that given a key \( K \in K \) and a ciphertext \( C \in C \) runs \( D(K, C) \), returning 0 if the result is \( \perp \) and 1 otherwise. We now formally define two notions of integrity. (We use the term integrity synonymously with authenticity.) Let \( \text{TYPE} \in \{\text{INT-PTXT}, \text{INT-CTXT}\} \); \( \text{INT-PTXT} \) denotes a notion of plaintext integrity of AE schemes and \( \text{INT-CTXT} \) denotes ciphertext integrity. Let \( A \) be an adversary with access to encryption oracle \( \text{enc} (\cdot) \) and verification oracle \( \text{vf} (\cdot) \). Similarly to MA schemes, we say that \( A \) is a \((t, q_s, q_v)\)-adversary if it runs in time at most \( t \), makes at most \( q_s \) encryption queries and at most
We define TYPE security of $\Pi$ via the security game depicted in Figure 2.5. We let
\[
\text{Adv}^\text{TYPE}_{\Pi}(A) = \Pr[A\text{TYPE sets win}]
\]
denote the winning probability for $A$ in the security game in question. We say that $\Pi$ is TYPE $\varepsilon$–secure against $(t, q_s, q_v)$–adversaries, if for any such adversary $A$, the advantage of $A$ in the corresponding security game, $\text{Adv}^\text{TYPE}_{\Pi}(A) < \varepsilon$.

In the security games, access to the encryption oracle allows the adversary to mount a chosen-message attack. The adversary succeeds if it makes the verification oracle accept an “illegitimate ciphertext.” Integrity of plaintexts calls a ciphertext illegitimate if the corresponding plaintext was never queried to the encryption oracle, while integrity of ciphertexts calls a ciphertext illegitimate if it was never returned by the encryption oracle. In each case one may consider either one or many verification queries. Historically, INT-PTXT and INT-CTXT under multiple verification queries are from [12] while INT-CTXT under single verification query is from [42, 14].

### 2.5.2 Results

**Theorem 2.5.1** Assume there exists AE scheme $\Pi$ which is INT-PTXT $\varepsilon$–secure against $(t, q_s, 1)$–adversaries. Then there exists another authenticated encryption scheme $\overline{\Pi}$, which is INT-PTXT $\varepsilon$–secure against $(t', q_s, 1)$–adversaries (for $t' = t - O(q_s)$) but which can be broken with probability 1 by an efficient multiple-queries adversary.

**Proof:** By assumption there exists a symmetric key encryption scheme $\Pi = (E, D)$ which is INT-PTXT secure against single-query adversaries. Let $K$ denote the key space for $\Pi$; without loss of generality we assume that $K \subset \{0, 1\}^l$ for some $l > 0$. The counterexample is the symmetric key encryption scheme $\overline{\Pi} = (\overline{E}, \overline{D})$ with the same key space $K$ whose encryption and decryption algorithms are specified below:
Algorithm $\mathcal{E}(K, M)$
\[
\begin{align*}
  c &\leftarrow \mathcal{E}(K, M) \\
  \overline{c} &\leftarrow (c, 0) \\
  \text{Return } \overline{c}
\end{align*}
\]

Algorithm $\mathcal{D}(K, \overline{c})$
\[
\begin{align*}
  \text{Parse } c \text{ as } (c, i) \text{ where } i \in \{0, 1, \ldots, l\} \\
  M &\leftarrow \mathcal{D}(K, c) \\
  \text{If } (M = \bot \text{ or } i = 0) \text{ then return } M \\
  \text{If } (M \neq \bot \text{ and } i \geq 1 \text{ and } K[i] = 1) \text{ then return } M \\
  \text{If } (M \neq \bot \text{ and } i \geq 1 \text{ and } K[i] = 0) \text{ then return } \bot
\end{align*}
\]

This encryption scheme has the following 2 properties:

1. There exists $(O(l), l, l + 1)$–adversary $\mathcal{A}$ such that
   \[
   \text{Adv}^{\text{INT-PTXT}}_{\Pi}(\mathcal{A}) = 1.
   \]

2. Let $\mathcal{A}$ be any polynomial-time INT-PTXT $(t', q_s, 1)$–adversary attacking $\Pi$. Then there exists INT-PTXT $(t, q_s, 1)$–adversary $\mathcal{A}$ attacking $\Pi$ such that
   \[
   \text{Adv}^{\text{INT-PTXT}}_{\Pi}(\mathcal{A}) \leq \text{Adv}^{\text{INT-PTXT}}_{\Pi}(\mathcal{A}).
   \]

The proofs of these claims are analogous to the ones for MA schemes and thus we omit them.

### 2.6 Acknowledgements

Parts of this chapter appeared in the paper


The dissertation author was the primary investigator and author of this paper.
3 Data Authentication Primitive

3.1 Definitions

**Data Authentication Primitive.** In order to establish security of various protocols under multiple verification queries, we introduce a new primitive called the *data authentication primitive*, which generalizes message authentication, authenticated encryption and other primitives.

A data authentication primitive $\Pi = (\text{SGN}, \text{VF})$ over sets $K$, $M$, $T$ and $V$ consists of two algorithms. The signing algorithm $\text{SGN} : K \times M \rightarrow T$ can be stateful or randomized. The corresponding verification algorithm $\text{VF} : K \times V \rightarrow \{0, 1\}$ is stateless and deterministic. Intuitively, $K$, $M$ and $T$ represent sets of keys, messages and authentication tags respectively. A set $V$ represents a set of possible verification queries.

Security of $\Pi$ is evaluated with respect to a deterministic *freshness function* $\text{Fresh}$, which takes input a verification message $V \in V$ and a list of tuples $(M_1, T_1), \ldots, (M_t, T_t) \in M \times T$ ($t \geq 0$) and outputs one bit.

**Security of Data Authentication Primitives.** We associate to a data authentication primitive $\Pi = (\text{SGN}, \text{VF})$ over sets $K$, $M$, $T$ and $V$ and a freshness function $\text{Fresh}$ the security game $\text{UF}$ (UF stands for UnForgeability), which is illustrated in Figure 3.1. This game is played with an adversary $A$, who has access to oracles $\text{sig}(\cdot)$ and $\text{vf}(\cdot)$.
Game UF

\( K \leftarrow \mathbf{K}; i \leftarrow 0; \text{win} \leftarrow \text{false} \)

– On query \( M \) to \( \text{sig}(\cdot) \):

\( i \leftarrow i + 1; M_i \leftarrow M; T_i \leftarrow \text{SGN}(K, M) \)

Reply \( T_i \)

– On query \( V \) to \( \text{vf}(\cdot) \):

\( d \leftarrow \text{VF}(K, V) \)

If \( d = 1 \) and \( \text{Fresh}(V, M_1, T_1, \ldots, M_i, T_i) = 1 \) then \( \text{win} \leftarrow \text{true} \)

Reply \( d \)

Figure 3.1  Game defining security of the data authentication primitive \( \Pi = (\text{SGN}, \text{VF}) \) over sets \( \mathbf{K}, \mathbf{M}, \mathbf{T} \) and \( \mathbf{V} \) with freshness function \( \text{Fresh} \).

The advantage of the adversary \( A \) participating in the UF game with data authentication primitive \( \Pi = (\text{SGN}, \text{VF}) \) with freshness function \( \text{Fresh} \) is

\[
\text{Adv}_{\Pi,\text{Fresh}}^{\text{UF}}(A) = \Pr \left[ A^{\text{UF}} \text{ sets win} \right].
\]

CAPTURING THE STANDARD PRIMITIVES VIA THE DATA AUTHENTICATION PRIMITIVE. We observe that many standard cryptographic primitives can be viewed as data authentication primitives with appropriate related freshness functions, such that security of these primitives is equivalent to security of the corresponding data authentication primitives. Therefore, the results of this work (which apply to any data authentication primitive) apply to each of the standard primitives.

In Section 3.2 we show how cryptographic primitives such as MA schemes, AE schemes, AEAD schemes, XOR-tag scheme, etc. are translated to data authentication primitives.

REVEALING FUNCTION. Consider some data authentication primitive \( \Pi = (\text{SGN}, \text{VF}) \) over sets \( \mathbf{K}, \mathbf{M}, \mathbf{T} \) and \( \mathbf{V} \) with associated freshness function \( \text{Fresh} \). Let Reveal be a
Experiment \( \text{REV}_{\Pi, \text{Fresh}}(K, V, M_1, \ldots, M_t) \)

\[
T_1 \leftarrow \text{SGN}(K, M_1); \ldots; T_t \leftarrow \text{SGN}(K, M_t)
\]

If \( \text{Fresh}(M_1, T_1, \ldots, M_t, T_t) = 0 \) and \( \text{Reveal}(V, M_1, T_1, \ldots, M_t, T_t) \neq \text{VF}(K, V) \)

return 0

Otherwise, return 1

Figure 3.2 Experiment defining a revealing function \( \text{Reveal} \) of the data authentication primitive \( \Pi = (\text{SGN}, \text{VF}) \) with freshness function \( \text{Fresh} \).

deterministic function which takes input a verification message \( V \in V \) and a list of tuples \( (M_1, T_1), \ldots, (M_t, T_t) \in M \times T \) \( (t \geq 0) \) and outputs one bit. Informally, \( \text{Reveal} \) is a revealing function for the data authentication primitive \( \Pi \) if it allows to answer those verification queries which are not fresh without calling the verification oracle.

Consider the experiment \( \text{REV}_{\Pi, \text{Fresh}}(\cdot) \) depicted in Figure 3.2, which is associated to \( \Pi, \text{Fresh} \) and \( \text{Reveal} \). We say that \( \text{Reveal} \) is a revealing function for the data authentication primitive \( \Pi \) with freshness function \( \text{Fresh} \) if for any \( K \in K \), for any \( V \in V \), for any \( t \geq 0 \) and for any \( t \) messages \( M_1, \ldots, M_t \in M \)

\[
\Pr \left[ \text{REV}_{\Pi, \text{Fresh}}(K, V, M_1, \ldots, M_t) = 1 \right] = 1.
\]

This probability is taken over random coins used by the signing algorithm \( \text{SGN} \) in the experiment. Note that the experiment returns 1 when the following condition holds:

if \( \text{Fresh}(V, M_1, T_1, \ldots, M_t, T_t) = 0 \), then \( \text{Reveal}(V, M_1, T_1, \ldots, M_t, T_t) \neq \text{VF}(K, V) \).

Adversarial Resources. We find it useful to quantify resources available to adversaries against data authentication primitives. We say that \( \mathcal{A} \) is \((t, q_s, q_v)\)–adversary if \( \mathcal{A} \) runs in time at most \( t \), makes at most \( q_s \) signing queries and at most \( q_v \) verification queries. We set either of \( t, q_s \) or \( q_v \) to \( \infty \) if an adversary is unbounded in a use of that resource.
Adversary $S[A]^{\text{sig}(\cdot), \text{vf}(\cdot)}$:

Initialize $i \leftarrow 0$

Run $A^{\text{sig}(\cdot), \text{vf}(\cdot)}$ and handle its queries as follows

On query $M$ to $\text{sig}'(\cdot)$:

- $i \leftarrow i + 1$
- $M_i \leftarrow M$
- obtain $T_i \leftarrow \text{sig}(M)$
- Store $(S_i, T_i)$
- Return $T_i$ to $A$

On query $V$ to $\text{vf}'(\cdot)$:

- Compute $\text{Fresh}(V, M_1, T_1, \ldots, M_i, T_i)$
- Obtain $b \leftarrow \text{vf}(V)$
- Store $(S_i, T_i)$
- Return $b$ to $A$

Figure 3.3 Construction of adversary $S[A]$, which is used to define $T(t, q_s, q_v)$.

Also, we find it useful to define time needed to run a $(t, q_s, q_v)$-adversary and handle all the queries it makes. Consider any data authentication primitive $\Pi = (\text{SGN}, \text{VF})$ with freshness function $\text{Fresh}$ and revealing function $\text{Reveal}$. Consider any $(t, q_s, q_v)$-adversary $A$ against $\Pi$ and the corresponding adversary $S[A]$ depicted in Figure 3.3. We denote by $T(t, q_s, q_v)$ an upper bound on a running time of the algorithm $S[A]$ (among all $(t, q_s, q_v)$-adversaries $A$), where the running time of $S[A]$ includes time for computing $\text{Fresh}$ and $\text{Reveal}$ but excludes time for oracle calls to $\text{sig}(\cdot)$ and $\text{vf}(\cdot)$.

**Random Coins Used by the UF Game and by Adversaries.** Some of our results require specifying particular random coins used by the UF game and those used by adversaries. For any probabilistic algorithm $A$ we denote by $\text{Coins}_A$ the set containing all possible values of random coins for $A$. Wlog we assume that $\text{Coins}_A$ is finite and that random coins for $A$ are chosen uniformly from $\text{Coins}_A$. For any $r \in \text{Coins}_A$ we denote by $A[r]$ the algorithm $A$ which makes all its random choices based on $r$.

For any data authentication primitive $\Pi = (\text{SGN}, \text{VF})$ over sets $K, M, T$ and $V$ with freshness function $\text{Fresh}$ and for any integer $q_s$ we denote by $\text{Coins}_{\text{UF}}(q_s)$ the set of all possible values of random coins for the UF game against adversaries making no
more than \( q_s \) signing queries. In general, randomness used by the UF game consists of a random choice of a key from \( K \) and of random choices made by the signing algorithm; therefore \( \text{Coins}_{UF}(q_s) = K \times (\text{Coins}_{SGN})^q_s \). For any \( s \in \text{Coins}_{UF}(q_s) \) we denote by \( UF_s \) the UF game which makes all its random choices based on \( s \).

### 3.2 Data Authentication Primitives for Existing Protocols

#### 3.2.1 Message Authentication (MA) Schemes

Let \( \Pi = (\text{SGN}, \text{VF}) \) be any MA scheme with a key space \( K \), message space \( M \) and a tag space \( T \). Such an MA scheme itself is a data authentication primitive over sets \( K, M, T \) and \( V = M \times T \). \( \Pi \) can be associated with either one of the following freshness functions:

\[
\text{Fresh}_{MA-UF}((M, T), M_1, T_1, \ldots, M_t, T_t) = \begin{cases} 
0, & \text{if } M \in \{M_1, \ldots, M_t\}; \\
1, & \text{otherwise}.
\end{cases}
\]

\[
\text{Fresh}_{MA-SUF}((M, T), M_1, T_1, \ldots, M_t, T_t) = \begin{cases} 
0, & \text{if } (M, T) \in \{(M_1, T_1), \ldots, (M_t, T_t)\}; \\
1, & \text{otherwise}.
\end{cases}
\]

Security of the data authentication primitive \( \Pi \) with freshness functions \( \text{Fresh}_{MA-UF} \) or \( \text{Fresh}_{MA-SUF} \) is equivalent to standard or strong unforgeability of the MA scheme \( \Pi \) respectively. Any adversary \( A \) against standard or strong unforgeability of the MA scheme is also an adversary against the data authentication primitive \( \Pi \) with respective freshness function \( \text{Fresh}_{MA-UF} \) or \( \text{Fresh}_{MA-SUF} \) such that

\[
\text{Adv}^{\text{MA-UF}}_{\Pi}(A) = \text{Adv}^{\Pi, \text{Fresh}_{MA-UF}}(A) \quad \text{and} \quad \text{Adv}^{\text{MA-SUF}}_{\Pi}(A) = \text{Adv}^{\Pi, \text{Fresh}_{MA-SUF}}(A).
\]

Here \( \text{MA-UF} \) and \( \text{MA-SUF} \) denote experiments defining respectively standard and strong unforgeability of MA schemes, as defined in Section 2.1.
Data authentication primitive \( \Pi \) with freshness function \( \text{Fresh}_{\text{MA-SUF}} \) has trivial revealing function

\[
\text{Reveal}_{\text{MA-SUF}}(V, M_1, T_1, \ldots, M_t, T_t) = 1.
\]

Consider any \( K \in K \), any verification message \( V = (M, T) \in V \), any \( t \) messages \( M_1, \ldots, M_t \in M \) and any tags \( T_1, \ldots, T_t \) computed in the experiment

\[
\text{REV}_{\Pi,\text{Fresh}_{\text{MA-SUF}}}(K, V, M_1, \ldots, M_t).
\]

If a verification message \( (M, T) \) is not fresh then there exists \( i \in \{1, \ldots, t\} \) such that \( M = M_i \) and \( T = T_i \). That is, \( \text{VF}(K, V) = \text{VF}(K, (M_i, T_i)) \) must return 1, as \( (M_i, T_i) \) is a valid message/tag pair. Therefore, if \( V \) is not fresh then \( \text{VF}(K, V) = 1 \) and \( \text{Reveal}_{\text{MA-SUF}} \) is indeed the revealing function for \( \Pi \) and \( \text{Fresh}_{\text{MA-SUF}} \).

On the other hand, in Section 3.4.1 we show that MA schemes with the freshness function \( \text{Fresh}_{\text{MA-UF}} \) do not necessarily have efficiently computable revealing functions. We show that a MA scheme from Section 2.2 doesn’t have an efficient revealing function under the assumption that secure MA schemes exist.

### 3.2.2 Authenticated Encryption (AE) Schemes

Let \( \Lambda = (E, D) \) be any authenticated encryption scheme with a key space \( K \), message space \( M \) and a ciphertext space \( C \). Such an AE scheme is transformed to the data authentication primitive \( \Pi[\Lambda] = (E, \text{VF}_\Lambda) \) over sets \( K, M, T = C \) and \( V = C \) with verification function

\[
\text{VF}_\Lambda(K, V) = \begin{cases} 
1, & \text{if } D_K(V) \neq \bot; \\
0, & \text{otherwise.}
\end{cases}
\]

We associate to \( \Pi[\Lambda] \) the freshness function

\[
\text{Fresh}_{\Lambda}(V, M_1, T_1, \ldots, M_t, T_t) = \begin{cases} 
0, & \text{if } V \in \{T_1, \ldots, T_t\}; \\
1, & \text{otherwise.}
\end{cases}
\]
Authenticity (integrity of ciphertexts) of the encryption scheme $\Lambda$ as defined in [12] is equivalent to security of the data authentication primitive $\Pi[\Lambda]$ with freshness function $\text{Fresh}_{\text{AE}}$. Any adversary $A$ against authenticity of the encryption scheme $\Lambda$ is also an adversary against the data authentication primitive $\Pi[\Lambda]$ and

$$\text{Adv}_{\Lambda}^{\text{INT-CTX}}(A) = \text{Adv}_{\Pi[\Lambda],\text{Fresh}_{\text{AE}}}(A).$$

$\Pi[\Lambda]$ with $\text{Fresh}_{\text{AE}}$ has trivial revealing function

$$\text{Reveal}_{\text{AE}}(V, M_1, T_1, \ldots, M_t, T_t) = 1.$$ 

Consider any $K \in \mathbf{K}$, any verification message $V \in \mathbf{V}$, any $t$ messages $M_1, \ldots, M_t \in \mathbf{M}$ and any tags $T_1, \ldots, T_t$ computed in the experiment

$$\text{REV}_{\Pi[\Lambda],\text{Fresh}_{\text{AE}}}(K, V, M_1, \ldots, M_t).$$

If a verification message $V$ is not fresh then $V = T_i$ for some $i \in \{1, \ldots, t\}$. Therefore $D_K(T_i) = M_i \neq \bot$ and $\text{VF}_\Lambda(V) = 1$. Thus, $\text{Reveal}_{\text{AE}}$ is indeed the revealing function for $\Pi[\Lambda]$.

We remark that integrity of plaintexts ($\text{INT-PTXT}$), another authenticity definition from [12], cannot be naturally expressed by a data authentication primitive. For AE schemes under integrity of plaintexts, the corresponding freshness function would also depend on the secret key selected in the UF experiment (and thus, the freshness function would not be publicly computable), which is not allowed by the definition of freshness function. We also remark that results of this work do not hold for AE schemes under integrity of plaintexts. As we showed in Section 2.5, $\text{INT-PTXT}$ security under one verification query does not imply $\text{INT-PTXT}$ security under multiple queries.

### 3.2.3 XOR-tag Schemes

XOR-tag schemes are defined in [15] and are used to establish security of the EAX mode. Let $I$ denote the set of all binary strings of length at most $m$ and let $c$ denote the size of $I$. Let $x_1, \ldots, x_c$ denote a lexicographic ordering of $I$. For each set $S \subset I$ we
let \( ChV(S) \in \{0, 1\}^c \) denote its \( c \)-bit characteristic vector, meaning \( ChV(S)[j] = 1 \) if \( x_j \in S \) and 0 otherwise (\( 1 \leq j \leq c \)). XOR-tag scheme \( \Pi[m, \tau] = (XTag, XVF) \) consists of the following two algorithms. Algorithm \( XTag \) takes input a function \( F : I \rightarrow \{0, 1\}^\tau \) and a subset \( S \subseteq I \) and returns

\[
XTag_F(S) = \bigoplus_{x \in S} F(x).
\]

Algorithm \( XVF \) takes input a function \( F : I \rightarrow \{0, 1\}^\tau \), a subset \( S \subseteq I \) and a tag \( T \in \{0, 1\}^\tau \) and returns 1 if \( T = XTag_F(S) \) and 0 otherwise.

In XOR-tag game, a function \( F : I \rightarrow \{0, 1\}^\tau \) is chosen at random and an adversary is given access to oracles \( XTag_F(\cdot) \) and \( XVF_F(\cdot, \cdot) \). The adversary is required to produce a set \( S \subseteq I \) and a tag \( T \in \{0, 1\}^\tau \) such that \( XVF_F(S, T) = 1 \) and \( ChV(S) \) is not a XOR of characteristic vectors of any previous \( XTag_F \) queries.

XOR-tag scheme \( \Pi[m, \tau] = (XTag, XVF) \) is the data authentication primitive over the following sets \( K, M, T \) and \( V \) with the following freshness function \( \text{Fresh}_{\text{XOR}} \):

\[ K \text{ is the set of all functions } I \rightarrow \{0, 1\}^\tau; \text{ a key } F \text{ is a random function } I \rightarrow \{0, 1\}^\tau \]
\[ M = 2^I, \text{ a message } S \text{ is a subset } S \subseteq I \]
\[ T = \{0, 1\}^\tau \]
\[ V = M \times T \]

\[
\text{Fresh}_{\text{XOR}}((S, T), S_1, T_1, \ldots, S_t, T_t) = \begin{cases} 
0, & \text{ if } \exists \{i_1, \ldots, i_\ell\} \subset \{1, \ldots, t\} \text{ s.t. } \\
ChV(S) = ChV(S_{i_1}) \oplus \ldots \oplus ChV(S_{i_\ell}); \\
1, & \text{ otherwise.}
\end{cases}
\]

Security of XOR-tag scheme with parameters \( m \) and \( \tau \) is equivalent to security of the data authentication primitive \( \Pi[m, \tau] \). Any adversary \( \mathcal{A} \) against XOR-tag game is also an adversary against the corresponding data authentication primitive \( \Pi[m, \tau] \) and

\[
\text{Adv}_{m, \tau}^{\text{XOR-TAG}}(\mathcal{A}) = \text{Adv}_{\Pi[m, \tau], \text{Fresh}_{\text{XOR}}}(\mathcal{A}).
\]
We observe that $\Pi[m, \tau]$ with $\text{Fresh}_{\text{XOR}}$ has the following revealing function:

\[
\text{Reveal}_{\text{XOR}}((M, T), M_1, T_1, \ldots, M_t, T_t) = \begin{cases} 
1, & \text{if } \exists \ L \subset \{1, \ldots, t\} \text{ s.t. } 
ChV(M) = \bigoplus_{i \in L} ChV(M_i) \text{ and } 
T = \bigoplus_{i \in L} T_i; \\
0, & \text{otherwise.}
\end{cases}
\]

Consider any function $F : I \to \{0, 1\}^\tau$, any verification message $V = (M, T) \in V$, any $t$ messages $M_1, \ldots, M_t \in M$ and any tags $T_1, \ldots, T_t$ computed in the experiment $\text{REV}_{\Pi[m, \tau], \text{Fresh}_{\text{XOR}}}(K, V, M_1, \ldots, M_t)$. If verification message $(M, T)$ is not fresh then $ChV(M) = ChV(M_{i_1}) \oplus \ldots \oplus ChV(M_{i_l})$ for some $\{i_1, \ldots, i_l\} \subset \{1, \ldots, t\}$. In this case, $\text{XTag}_F(M) = \text{XTag}_F(M_{i_1}) \oplus \ldots \oplus \text{XTag}_F(M_{i_l})$. Therefore, $\text{XVF}_F(M, T) = 1$ if and only if $T = T_{i_1} \oplus \ldots \oplus T_{i_l}$.

### 3.2.4 Nonce-Based MA Schemes

Let $\Lambda = (\text{SGN}, \text{VF})$ be any nonce-based MA scheme with a key space $K$, a nonce space $N$, a message space $M'$ and a tag space $T'$. $\Lambda$ can be transformed to the data authentication primitive $\Pi[\Lambda] = (\text{SGN}_\Lambda, \text{VF})$ over sets $K, M = N \times M', T = T' \cup \{\bot\}$ and $V = N \times M' \times T'$ with the following signing algorithm $\text{SGN}_\Lambda$ and freshness function $\text{Fresh}_{\text{NMA}}$. Signing algorithm $\text{SGN}_\Lambda$ is stateful; the state is a set $S \subset N$, which is initially empty.

\[
\text{SGN}_\Lambda(K, (N, M)) = \begin{cases} 
\text{SGN}(K, N, M), & \text{update } S \leftarrow S \cup \{N\}, \text{ if } N \not\in S; \\
\bot, & \text{otherwise.}
\end{cases}
\]

\[
\text{Fresh}_{\text{NMA}}((N, M, T), (N_1, M_1), T_1, \ldots, (N_t, M_t), T_t) = 
\begin{cases} 
0, & \text{if } ((N, M), T) \in \{(N_1, M_1), T_1\}, \ldots, ((N_t, M_t), T_t); \\
1, & \text{otherwise.}
\end{cases}
\]

Security of nonce-based MA scheme $\Lambda$ is equivalent to security of data authentication primitive $\Pi[\Lambda]$. Any adversary $A$ against $\Lambda$ is also an adversary against $\Pi[\Lambda]$ and

\[
\text{Adv}_\Lambda^{\text{MA}}(A) = \text{Adv}_{\Pi[\Lambda], \text{Fresh}_{\text{NMA}}}(A).
\]
It is not hard to show that \( \Pi[\Lambda] \) with \( \text{Fresh}_{\text{NMA}} \) has trivial revealing function

\[
\text{Reveal}_{\text{NMA}}(V, (N_1, M_1), T_1, \ldots, (N_t, M_t), T_t) = 1.
\]

\( \Pi[\Lambda] \) with \( \text{Fresh}_{\text{NMA}} \) only captures strong unforgeability for nonce-based MA schemes, as defined in [17]. One can define a freshness function for \( \Pi[\Lambda] \), which will capture standard unforgeability similarly to the case of regular MA schemes. We do not discuss in detail nonce-based MA schemes with standard unforgeability as they violate our theorems.

### 3.2.5 Authenticated Encryption with Associated Data (AEAD)

Let \( \Lambda = (E, D) \) be any AEAD scheme with a key space \( K \), nonce space \( N \), header space \( H \), message space \( M' \) and a ciphertext space \( C \), as defined in [57]. AEAD scheme \( \Lambda \) can be transformed to the data authentication primitive \( \Pi[\Lambda] = (\text{SGN}_{\Lambda}, \text{VF}_{\Lambda}) \) over sets \( K, M = N \times H \times M', T = C \cup \{\bot\} \) and \( V = N \times H \times M' \times C \) with freshness function \( \text{Fresh}_{\text{AEAD}} \) as follows. Signing algorithm \( \text{SGN}_{\Lambda} \) is stateful; the state is a set \( S \subset N \), which is initially empty.

\[
\begin{align*}
\text{SGN}_{\Lambda}(K, (N, H, M)) &= \begin{cases} 
E_K(N, H, M), & \text{update } S \leftarrow S \cup \{N\}, \text{ if } N \not\in S; \\
\bot, & \text{otherwise.}
\end{cases} \\
\text{VF}_{\Lambda}(K, (N, H, M, C)) &= \begin{cases} 
1, & \text{if } D_K(N, H, C) \neq \bot; \\
\bot, & \text{otherwise.}
\end{cases}
\end{align*}
\]

\[
\text{Fresh}_{\text{AEAD}}((N, H, M, C), (N_1, H_1, M_1), T_1, \ldots, (N_t, H_t, M_t), T_t) = \\
\begin{cases} 
0, & \text{if } ((N, H, M), C) \in \{((N_1, H_1, M_1), T_1), \ldots, ((N_t, H_t, M_t), T_t)\}; \\
1, & \text{otherwise.}
\end{cases}
\]

Security of AEAD scheme \( \Lambda [57] \) is equivalent to security of data authentication primitive \( \Pi[\Lambda] \). Any adversary \( A \) against \( \Lambda \) is also an adversary against \( \Pi[\Lambda] \) and

\[
\text{Adv}_{\Lambda}^{\text{AEAD}}(A) = \text{Adv}_{\Pi[\Lambda], \text{Fresh}_{\text{AEAD}}}^\text{UF}(A).
\]

It is not hard to show that \( \Pi[\Lambda] \) with freshness function \( \text{Fresh}_{\text{AEAD}} \) has trivial revealing function

\[
\text{Reveal}_{\text{AEAD}}(V, (N_1, H_1, M_1), T_1, \ldots, (N_1, H_1, M_1), T_t) = 1.
\]
We observe that nonce-based AE schemes can be viewed as a particular case of AEAD schemes with the empty headers. That is why nonce-based AE schemes can be transformed to data authentication primitives using the same construction as AEAD schemes.

3.3 Any Revealing Function Is Equivalent to Trivial

As we demonstrated in Section 3.2, for many data authentication primitives the revealing function is trivial (always returns 1). We observe that this behavior is not surprising. In fact, any revealing function is equivalent to trivial in the following sense.

**Proposition 3.3.1** Let \( \Pi = (\text{SGN}, \text{VF}) \) be any data authentication primitive over sets \( K, M, T, V \) with freshness function \( \text{Fresh} \) and revealing function \( \text{Reveal} \). Then there exists another freshness function \( \text{Fresh}^* \) such that \( \Pi \) with \( \text{Fresh}^* \) has trivial revealing function \( \text{Reveal}^*(\ldots) = 1 \) and such that for any adversary \( A \)

\[
\text{Adv}^{\text{UF}}_{\Pi, \text{Fresh}}(A) = \text{Adv}^{\text{UF}}_{\Pi, \text{Fresh}^*}(A). \tag{3.1}
\]

**Proof:** The desired freshness function \( \text{Fresh}^* \) can be constructed as follows:

\[
\text{Fresh}^*(V, M_1, T_1, \ldots, M_t, T_t) = \begin{cases} 
0, & \text{if } \text{Fresh}(V, M_1, T_1, \ldots, M_t, T_t) = 0 \\
& \text{and Reveal}(V, M_1, T_1, \ldots, M_t, T_t) = 1; \\
1, & \text{otherwise}.
\end{cases}
\]

Consider any key \( K \in K \), any verification message \( V \in V \), any \( t \) messages \( M_1, \ldots, M_t \in M \) and any tags \( T_1, \ldots, T_t \) computed in the game \( \text{RE}^{\Pi, \text{Fresh}^*}(K, V, M_1, \ldots, M_t) \) associated to \( \text{Reveal}^* \). Let \( \sigma \) denote the list \((V, M_1, T_1, \ldots, M_t, T_t)\). By the construction of \( \text{Fresh}^* \), if \( \text{Fresh}^*(\sigma) = 0 \) then \( \text{Fresh}(\sigma) = 0 \) and \( \text{Reveal}(\sigma) = 1 \). The definition of the revealing function implies that in this case \( \text{VF}(K, V) = 1 \). Therefore, the trivial revealing function \( \text{Reveal}^* \) is a revealing function for \( \Pi \) with \( \text{Fresh}^* \).

Now let’s prove Equation (3.1). Consider any adversary \( A \) who participates in the UF game with \( \Pi \) and either \( \text{Fresh} \) or \( \text{Fresh}^* \). The choice of a freshness function influences
only the winning conditions of the game (that is, setting the variable \textit{win}), while signing and verification oracles in both games are handled identically. We observe that winning condition in both games are also identical. Consider any key $K \in \mathcal{K}$ and assume that $A$ made $i$ signing queries $M_1, \ldots, M_i$, received from the signing oracle $i$ tags $T_1, \ldots, T_i$ and that $A$ is making a verification query $V$. Denote the list $(V, M_1, T_1, \ldots, M_i, T_i)$ by $\sigma$. The first game sets \textit{win} if $\text{VF}(K, V) = 1$ and $\text{Fresh}(\sigma) = 1$. The second game sets \textit{win} if $\text{VF}(K, V) = 1$ and $\text{Fresh}^\ast(\sigma) = 1$. By the construction of $\text{Fresh}^\ast$, $\text{Fresh}^\ast(\sigma) = 1$ if either $\text{Fresh}(\sigma) = 1$ or both $\text{Fresh}(\sigma) = 0$ and $\text{Reveal}(\sigma) = 0$. The only difference between these winning conditions is that the second game sets \textit{win} when $\text{VF}(K, V) = 1$, $\text{Fresh}(\sigma) = 0$ and $\text{Reveal}(\sigma) = 0$. We note that this case never happens: by the definition of a revealing function, when $\text{Fresh}(\sigma) = 0$, $\text{Reveal}(\sigma) = \text{VF}(K, V)$ and therefore the case when $\text{VF}(K, V) = 1$ and $\text{Reveal}(\sigma) = 0$ never happens. We just showed that the winning conditions in both games are identical and thus for any adversary $A$

$$\text{Adv}_{\Pi,\text{Fresh}}^\text{UF}(A) = \text{Adv}_{\Pi,\text{Fresh}^\ast}^\text{UF}(A).$$

## 3.4 Security of Data Authentication Primitives against Multiple Verification Queries

### 3.4.1 Security under One Query Implies Security under Multiple Queries

\textbf{Lemma 3.4.1} Assume that a data authentication primitive $\Pi = (\text{SGN}, \text{VF})$ over sets $\mathcal{K}, \mathcal{M}, \mathcal{T}, \mathcal{V}$ with freshness function $\text{Fresh}$ has revealing function $\text{Reveal}$. Then, for any $(t, q_s, q_v)$--adversary $A$ against $\Pi$ with $\text{Fresh}$, there exists $(t', q_s, q_v)$--adversary $B$ s.t. $t' = T(t, q_v, q_s)$, all $B$'s verification queries are fresh and

$$\text{Adv}_{\Pi,\text{Fresh}}^\text{UF}(A) = \text{Adv}_{\Pi,\text{Fresh}}^\text{UF}(B).$$
Adversary $B$:

\[ i \leftarrow 0; \text{run } \mathcal{A} \text{ and handles } \mathcal{A}' \text{’s queries as follows:} \]

**Signing query** $M$:

\[ i \leftarrow i + 1; M_i \leftarrow M; T_i \leftarrow \text{SGN}(M) \]

Reply $T_i$

**Verification query** $V$:

If $\text{Fresh}(V, M_1, T_1, \ldots, M_i, T_i) = 0$ then $d \leftarrow \text{Reveal}(V, M_1, T_1, \ldots, M_i, T_i)$

Otherwise $d \leftarrow \text{VF}(V)$

Reply $d$

---

**Figure 3.4** Construction of the adversary $B$.

---

**Proof:** Let $\mathcal{A}$ be any $(t, q_s, q_v)$–adversary. Consider the adversary $B$ which is described in Figure 3.4. The running time of $B$ is that of $\mathcal{A}$ plus time needed to handle oracle queries made by $\mathcal{A}$. $B$ handles each signing query in constant time and on each verification query, it computes $\text{Fresh}$ and $\text{Reveal}$ functions on inputs of length $\leq q_s$.

$B$ queries to the oracles the same signing queries as $\mathcal{A}$ and only those of $\mathcal{A}$’s verification queries which are fresh. For non-fresh verification queries, $B$ uses the revealing function to answer them. By the definition of revealing function, $B$’s answers to these verification queries are the same as true answers of the verification oracle. That is, all replies to $\mathcal{A}$ provided by $B$ are distributed exactly as oracle replies to $\mathcal{A}$ in the original UF game.

$\mathcal{A}$ wins the UF game against $\Pi$ with $\text{Fresh}$ if it makes a fresh verification query which is accepted by $\text{VF}$. If $\mathcal{A}$ eventually makes such a query, then $B$ makes the same query to $\text{VF}$ and wins the UF game. Thus, $B$ wins the UF game with the same probability as $\mathcal{A}$ does. \[\square\]

**Theorem 3.4.2** Assume that a data authentication primitive $\Pi = (\text{SGN}, \text{VF})$ over sets $\mathcal{K}, \mathcal{M}, \mathcal{T}, \mathcal{V}$ with freshness function $\text{Fresh}$ has revealing function $\text{Reveal}$. Then for any
Adversary $S(i, r)$

Initialize a counter $CTR \leftarrow 1$

Run $A_{[r]}$ and reply to its queries as follows

Signing query $M$:
- Obtain $T \leftarrow \text{SGN}(M)$, return $T$ to $A_{[r]}$

Verification query $V$:
- If $CTR \neq i$, update $CTR \leftarrow CTR + 1$ and return 0
- If $CTR = i$, call $VF(V)$ and halt

Adversary $B$

Pick $i \leftarrow \{1, \ldots, q_v\}$, $r \leftarrow \text{Coins}_A$

Execute $S(i, r)$

Figure 3.5 Construction of algorithms $S$ and $B$.

(t, $q_s, q_v$)–adversary $A$ there exists $(t', q_s, 1)$–adversary $B$ such that $t' = T(t, q_v, q_s)$ and

$$\text{Adv}^{\text{UF}}_{\Pi, \text{Fresh}}(B) \geq \frac{1}{q_v} \text{Adv}^{\text{UF}}_{\Pi, \text{Fresh}}(A).$$

**Proof:** Consider any data authentication primitive $\Pi = (\text{SGN}, VF)$ over sets $K, M, T, V$ with freshness function $\text{Fresh}$ and revealing function $\text{Reveal}$ and any adversary $A$ against $\Pi$. Using Lemma 3.4.1 we can assume that all verification queries made by $A$ are fresh.

Consider algorithms $S$ and $B$ depicted in Figure 3.5. We claim that $B$ is the desired adversary. Clearly, $B$ makes a single verification query and runs in the specified time. Let’s bound the success probability of $B$.

Recall the notation for random coins of $A$ and the UF game from Section 3.1. Fix any $s \in \text{Coins}_{\Pi}(q_s)$ and consider the game $\Pi_{[s]}$. This game is deterministic: the same key $K \in K$ is selected and signing queries are answered deterministically. Fix any
Consider executions of $A_{[r]}$, $S(1, r), \ldots, S(q_v, r)$ participating in the game $UF_{[s]}$.

The key observation is that if $A_{[r]}$ wins the game $UF_{[s]}$, then so does one of $S(i, r)$-s. Specifically, the winning algorithm is $S(i^*, r)$, where $i^*$ is an index of the first verification query made by $A_{[r]}$ which returned 1. Recall that $S(i^*, r)$ runs $A_{[r]}$ and replies with 0-s to first $i^*-1$ verification queries made by $A_{[r]}$. Inputs received by $A_{[r]}$ which is executed by $S(i^*, r)$ up to $i^*$-th verification query are the same as the ones received by $A_{[r]}$ in the true game $UF_{[s]}$. Thus, $i^*$-th verification query as well as all prior signing queries made by $A_{[r]}$ which is run by $S(i^*, r)$ are the same as the corresponding queries made by $A_{[r]}$ in the true game $UF_{[s]}$. Therefore, if $i^*$-th verification query made by $A_{[r]}$ in the game $UF_{[s]}$ is fresh and returns 1 then so is the verification query made by $S(i^*, r)$.

Denote by $Win_A \subset Coins_A \times Coins_{UF}(q_s)$ the set of pairs $(r, s)$ such that $A_{[r]}$ wins the game $UF_{[s]}$. Note that the winning probability of $A$ is

$$\text{Adv}_{UF, \text{Fresh}}^A = \Pr_{(r, s) \in Coins_A \times Coins_{UF}(q_s)} \left[ A_{[r]} \text{ wins } UF_{[s]} \right]$$

$$= \frac{|Win_A|}{|Coins_A| \cdot |Coins_{UF}(q_s)|}. \quad (3.2)$$

Similarly, for each $i \in \{1, \ldots, q_v\}$ denote by $Win_{S,i} \subset Coins_A \times Coins_{UF}(q_s)$ the set of pairs $(r, s)$ such that $Exp_{UF, \text{Fresh}}(S(i, r))_{[s]}$. Note that the winning probability of $B$ is equal to

$$\text{Adv}_{UF, \text{Fresh}}^B = \frac{1}{q_v} \cdot \sum_{i=1}^{q_v} \Pr_{(r, s) \in Coins_A \times Coins_{UF}(q_s)} \left[ S(i, r) \text{ wins } UF_{[s]} \right]$$

$$= \frac{1}{q_v} \cdot \frac{1}{|Coins_A| \cdot |Coins_{UF}(q_s)|} \cdot \sum_{i=1}^{q_v} |Win_{S,i}|. \quad (3.3)$$

Above we showed that for any $r, s \in \{0, 1\}^l$ if $A_{[r]}$ wins $UF_{[s]}$, then so does one of $S(i, r)$-s. This implies that

$$Win_A \subset (Win_{S,1} \cup \ldots \cup Win_{S,1}). \quad (3.4)$$

Applying to (3.6) the union bound, we have the following inequality:

$$|Win_A| \leq |Win_{S,1}| + \ldots + |Win_{S,1}|. \quad (3.5)$$
Combining inequality (3.7) with expressions for advantages of $\mathcal{A}$ and $\mathcal{B}$ from (3.3) and (3.5) we obtain that

$$\text{Adv}_{\Pi, \text{Fresh}}^{\text{UF}}(\mathcal{A}) \leq q_v \cdot \text{Adv}_{\Pi, \text{Fresh}}^{\text{UF}}(\mathcal{B}).$$

Theorem 3.4.2 establishes security against multiple verification queries of all data authentication primitives from Section 3.2 that have revealing functions. We find it useful to state this result separately for $\text{MA-SUF}$ security of MA schemes. Applying Theorem 3.4.2 to MA schemes with freshness function $\text{Fresh}_{\text{MA-SUF}}$ we get the following corollary.

**Corollary 3.4.3** Consider arbitrary MA scheme $\Pi = (\text{SGN}, \text{VF})$ and arbitrary integers $t$, $q_s$ and $q_v$. Then for any $(t, q_s, q_v)$--adversary $\mathcal{A}$ against strong unforgeability of $\Pi$ there exists another $(t', q_s, 1)$--adversary $\mathcal{B}$ such that $t' = t + O(q_s + q_v)$ and

$$\text{Adv}_{\Pi, \text{MA-SUF}}^{\text{MA-SUF}}(\mathcal{B}) \geq \frac{1}{q_v} \text{Adv}_{\Pi, \text{MA-SUF}}^{\text{MA-SUF}}(\mathcal{A}).$$

Similarly to the above corollary which establishes security of MA schemes one can state equivalent corollaries which establish security of all the other cryptographic primitives from Section 3.2 (for which corresponding data authentication primitives have revealing functions).

As showed in Section 2.2, for standard unforgeability of MA schemes security against one verification query does not imply security against multiple queries (under assumption that $\text{MA-UF}$ secure MA schemes exist). There we presented a construction of MA scheme $\Pi$, which is $\text{MA-UF}$ secure against adversaries making 1 verification query but is insecure against adversaries making multiple verification queries. Applying Theorem 3.4.2 to the data authentication primitive $\Pi$ we obtain the following.

**Corollary 3.4.4** Assuming the existence of $\text{MA-UF}$ secure MA scheme $\Pi$, there exists a MA scheme $\Pi'$ such that data authentication primitive $\Pi'$ with freshness function
Fresh_{MA-UF} doesn’t have an efficient revealing function. Specifically, if $\Pi$ is $\varepsilon$–secure against $(t, q_s, 1)$–adversaries and has key space $K \subset \{0, 1\}^l$ (such that $l \leq 1/\varepsilon - 1$), $\Pi$ doesn’t have a revealing function which is computable in time $\omega(t/l)$ on inputs of length at most $q_s$.

3.4.2 Improving Concrete Security

Consider some data authentication primitive $\Pi = (\text{SGN}, VF)$ over sets $K, M, T$ and $V$ with associated freshness function Fresh. Consider a class of adversaries against $\Pi$ with Fresh who make no more than $q_s$ signing queries. Recall that the game UF (played with adversaries from this class) uses random coins from the set $Coins_{UF}(q_s)$.

**Theorem 3.4.5** Assume that a data authentication primitive $\Pi = (\text{SGN}, VF)$ over sets $K, M, T, V$ with freshness function Fresh has revealing function Reveal. Consider any integers $t, q_s$ and $q_v$. Let $\text{Hard} : Coins_{UF}(q_s) \to \{0, 1\}$ be any function which takes input random coins used in the UF game and outputs one bit. Let $Ev_{\text{Hard}}$ be an event that the UF game selects random coins $s$ such that $\text{Hard}(s) = 1$. Assume that

$$\Pr[\neg Ev_{\text{Hard}}] = \Pr_{s \leftarrow Coins_{UF}(q_s)}[\text{Hard}(s) = 0] \leq \varepsilon_1.$$

Assume that for any $(t', q_s, 1)$–adversary $B$ (where $t' = T(t, q_v, q_s)$) against $\Pi$

$$\Pr[B_{UF} \text{ sets win } \land Ev_{\text{Hard}}] \leq \varepsilon_2.$$

Then for any $(t, q_s, q_v)$–adversary $A$

$$\text{Adv}_{\Pi, \text{Fresh}}^\text{UF}(A) \leq \varepsilon_1 + q_v \varepsilon_2.$$

**Proof:** Consider any $(t, q_s, q_v)$–adversary $A$; by Lemma 3.4.1 we can assume that all verification queries made by $A$ are fresh. Similarly to the proof of Theorem 3.4.2 consider algorithms $S$ and $B$ depicted in Figure 3.5.
Consider sets \( \text{Win}_A, \text{Win}_{S,1}, \ldots, \text{Win}_{S,q_v} \subset \text{Coins}_A \times \text{Coins}_{\text{UF}}(q_s) \) defined in the proof of Theorem 3.4.2. Consider a set

\[
\text{Hard} = \text{Coins}_A \times \{ s \in \text{Coins}_{\text{UF}}(q_s) \mid \text{Hard}(s) = 1 \}.
\]

Note that joint probabilities that \( A \) and \( B \) win the UF game and that the event \( \text{Ev}_{\text{Hard}} \) happens can be computed as follows:

\[
\begin{align*}
\Pr \left[ A \text{UF sets win} \land \text{Ev}_{\text{Hard}} \right] & = \frac{|\text{Win}_A \cap \text{Hard}|}{|\text{Coins}_A| \cdot |\text{Coins}_{\text{UF}}(q_s)|}; \\
\Pr \left[ B \text{UF sets win} \land \text{Ev}_{\text{Hard}} \right] & = \frac{1}{q_v} \sum_{i=1}^{q_v} \Pr \left[ \text{S}(i, r) \text{UF sets win} \land \text{Hard}(s) = 0 \right] \\
& = \frac{1}{q_v} \sum_{i=1}^{q_v} \frac{|\text{Win}_{S,i} \cap \text{Hard}|}{|\text{Coins}_A| \cdot |\text{Coins}_{\text{UF}}(q_s)|}.
\end{align*}
\]

(3.8) (3.9) (3.10)

Fix any \( r \in \text{Coins}_A \) and \( s \in \text{Coins}_{\text{UF}}(q_s) \). In the proof of Theorem 3.4.2 we showed that if \( A_r \) wins game \( \text{UF}_s \) then so does \( \mathcal{S}(i^*, r) \) for some \( i^* \in \{1, \ldots, q_v\} \) and therefore

\[
\text{Win}_A \subset \text{Win}_{S,1} \cup \ldots \cup \text{Win}_{S,q_v}.
\]

(3.11)

Intersecting both right-hand side and left-hand side of (3.11) with \( \text{Hard} \) we obtain that

\[
(\text{Win}_A \cap \text{Hard}) \subset (\text{Win}_{S,1} \cap \text{Hard}) \cup \ldots \cup (\text{Win}_{S,q_v} \cap \text{Hard}).
\]

(3.12)

By applying the union bound to (3.12), we have the following inequality:

\[
|\text{Win}_A \cap \text{Hard}| \leq |\text{Win}_{S,1} \cap \text{Hard}| + \ldots + |\text{Win}_{S,q_v} \cap \text{Hard}|.
\]

(3.13)

Combining (3.13) with (3.8) and (3.10) we obtain that

\[
\Pr \left[ A \text{UF sets win} \land \text{Ev}_{\text{Hard}} \right] \leq q \cdot \Pr \left[ B \text{UF sets win} \land \text{Ev}_{\text{Hard}} \right].
\]

Finally we can bound the advantage of \( A \) as follows:

\[
\text{Adv}_{\Pi, \text{Fresh}}(A) = \Pr \left[ A \text{UF sets win} \land \neg \text{Ev}_{\text{Hard}} \right] + \Pr \left[ A \text{UF sets win} \land \text{Ev}_{\text{Hard}} \right] \\
\leq \Pr \left[ \text{Ev}_{\text{Hard}} \right] + q \cdot \Pr \left[ B \text{UF sets win} \land \text{Ev}_{\text{Hard}} \right] \\
\leq \varepsilon_1 + q \cdot \varepsilon_2
\]
We observe that Theorem 3.4.5 allows to improve concrete security of data authentication primitives against multiple verification queries over what implied by Theorem 3.4.2. For some data authentication primitives there exist certain random coins for the UF game, which trivially allow an adversary to do the forgery. For example, some MA schemes involve nonces which are randomly selected by the signing algorithm. In this case, weak random coins are those which make some two nonces collide. If two nonces collide, it is usually easy for an adversary to do the forgery.

Function Hard is supposed to distinguish between random coins which allow for easy forgery \( \text{Hard}(s) = 0 \) and those which do not \( \text{Hard}(s) = 1 \). Event \( \neg\text{Ev}_{\text{Hard}} \) represents the unfortunate event that the UF game selects weak random coins. If

\[
\Pr[\neg\text{Ev}_{\text{Hard}}] \leq \varepsilon_1
\]

and for any \((t', q_s, 1)\)-adversary \( B \) \( \Pr[B^{\text{UF}} \text{ sets } \text{win} \land \text{Ev}_{\text{Hard}}] \leq \varepsilon_2 \), then advantage of any \((t', q_s, 1)\)-adversary can be bounded by \( \varepsilon_1 + \varepsilon_2 \). Theorem 3.4.2 implies that advantage of any \((t, q_s, q_v)\)-adversary is at most \( q_v \cdot (\varepsilon_1 + \varepsilon_2) \). Theorem 3.4.5 improves this bound to \( \varepsilon_1 + q_v \cdot \varepsilon_2 \), which is useful in analyzing concrete security of some data authentication primitives.

### 3.5 Reduction Tightness

Using Theorem 3.4.2 to establish security of data authentication primitives against multiple verification queries results in a factor \( q_v \) loss in the advantage. As we have discussed earlier, this can be appreciable in practice. This section looks into getting better reductions.

**No Better Reduction for Theorem 3.4.2.** The first, natural question to ask is whether Theorem 3.4.2 represents the best possible reduction, or whether there is a better one. We claim there is no better one, even for the special case of data authentication primitives that are MACs. To establish this we begin with the following:
Proposition 3.5.1 Assume there exist a PRF which is $\varepsilon$–secure against adversaries running in time $t$ and making $q$ oracle queries. Then for any integer $q_v$ there exists a MAC $\Pi$ such that

1. There exists an efficient adversary $A$ that makes at most $q_v$ verification queries and achieves $\text{Adv}_{\Pi}^{\text{MA-SUF}}(A) = 1$, and

2. For any $(t', q - 1, 1)$–adversary $B$, $\text{Adv}_{\Pi}^{\text{MA-SUF}}(B) \leq 1/q_v + \varepsilon$.

Proof: Let $F : K \times M \to \{1, \ldots, q_v\}$ be a PRF. Let $\Pi = \Pi[F] = (\text{SGN}, \text{VF})$ be the associated PRF-based MAC, with key space $K$, message space $M$ and tag space $\{1, \ldots, q_v\}$. (Note by definition $\text{SGN} = F$.) Adversary $A$ works as follows:

Adversary $A^{\text{sig}(\cdot), \text{vf}(\cdot, \cdot)}$

Pick arbitrary $M \in M$

For each $T \in \{1, \ldots, q_v\}$ do $d(T) \leftarrow \text{vf}(M, T)$

Since $\text{SGN}(K, M) = F(K, M) \in \{1, \ldots, q_v\}$ we have $\text{Adv}_{\Pi}^{\text{MA-SUF}}(A) = 1$. The number of verification queries made by $A$ is at most $q_v$. So we have item (1) of the Proposition. Now suppose $B$ is a $(t', q - 1, 1)$–adversary. We associate to it PRF adversary $D$ that, given an oracle $f(\cdot)$, works as follows. It runs $B$ on and when $B$ makes a query $M$ to its tagging oracle, $D$ returns $f(M)$ to $B$. When $B$ makes a query $(M, T)$ to its verification oracle, $D$ halts, outputting 1 if $f(M) = T$ and $M$ was not a query to the tagging oracle, and outputting 0 otherwise. Then

$$\Pr \left[ D^{F(K, \cdot)} = 1 \right] = \text{Adv}_{\Pi}^{\text{MA-SUF}}(B) \quad \text{and} \quad \Pr \left[ B^{f(\cdot)} = 1 \right] \leq \frac{1}{q_v},$$

where in the first case the probability is over a random choice of $K \in K$ and in the second over a random choice of function $f : M \to \{1, \ldots, q_v\}$. So we have

$$\text{Adv}_{\Pi}^{\text{MA-SUF}}(B) \leq \frac{1}{q_v} + \text{Adv}_{F}^{\text{PRF}}(D).$$

Since $D$ runs in time at most $t$ and makes at most $q$ oracle queries, $\text{Adv}_{F}^{\text{PRF}}(D) < \varepsilon$, and thus we have item (2) of the Proposition.

---

1Here $t' = t - O(q)$. 
Now let us see why this Proposition shows that the reduction of Theorem 3.4.2 cannot be improved. View a reduction as a transform $B$ that given a data authentication primitive $\Pi$ with freshness function $\text{Fresh}$ and a $(t, q_s, q_v)$–adversary $A$ outputs a $(t', q_s, 1)$–adversary $B(\Pi, \text{Fresh}, A)$, where $t' = T(t, q_s, q_v)$. We say that $B$ has reduction factor $\phi(\cdot)$ if

$$\text{Adv}_{\Pi, \text{Fresh}}^{\text{UF}}(B(\Pi, \text{Fresh}, A)) \geq \phi(q_v) \cdot \text{Adv}_{\Pi, \text{Fresh}}^{\text{UF}}(A)$$

for all $\Pi$, $\text{Fresh}$ and $A$. Theorem 3.4.2 provides a reduction with reduction factor $\phi(n) = 1/n$. Now we claim:

**Corollary 3.5.2** If $\varepsilon$–secure PRFs exist then the reduction factor of any reduction is at most $1/q_v + \varepsilon$.

**Proof:** Consider any integer $q_v$ and let $\Pi$ be the MAC associated to it by Proposition 3.5.1. $\Pi$ is a data authentication primitive with freshness function $\text{Fresh}_{\text{MA-SUF}}$. Let $B$ be a reduction with reduction factor $\phi$. Proposition 3.5.1 says there is an efficient adversary $A$, making $q_v$ verification queries, such that

$$\text{Adv}_{\Pi, \text{Fresh}_{\text{MA-SUF}}}^{\text{UF}}(B(\Pi, \text{Fresh}, A)) \leq \frac{1}{q_v} + \varepsilon = \left(\frac{1}{q_v} + \varepsilon\right) \cdot \text{Adv}_{\Pi, \text{Fresh}_{\text{MA-SUF}}}^{\text{UF}}(A).$$

So $\phi(q_v) < 1/q_v + \varepsilon$. 

So no reduction has a reduction factor that is better than the one of Theorem 3.4.2. One might observe that the MAC of Proposition 3.5.1 is not MA-SUF secure. This did not preclude it being useful with regard to establishing the un-improvability of the reduction of Theorem 3.4.2, for the data authentication primitive in that theorem is also not assumed to be secure in any sense. Indeed, a reduction transforms an adversary of one type into an adversary of another type with some relation between the advantages, regardless of whether the underlying scheme is secure or not, so the tightness of its reduction factor may be assessed via its performance on insecure schemes.
3.6 Establishing Security of Practical Protocols under Multiple Verification Queries

3.6.1 Classes of MA-SUF Secure MA Schemes

Towards applying Theorem 3.4.2 to existing MA schemes, we establish MA-SUF security for certain important classes.

MACs. Suppose MA scheme $\Pi = (\text{SGN}, \text{VF})$ over sets $K, M$ and $T$ is a MAC. This implies that for any key $K \in K$ and message $M \in M$ there is only one tag that VF will accept, namely $\text{SGN}(K, M)$. Thus:

**Proposition 3.6.1** Let $\Pi$ be a MA-UF $\varepsilon$–secure MAC against $(t, q_s, q_v)$–adversaries. Then it is also MA-SUF $\varepsilon$–secure against the same class of adversaries.

So by Theorem 3.4.2, any MAC which is MA-UF secure against single-query adversaries is MA-SUF secure against multiple-queries adversaries. As an application, since HMAC is only proven in [3] to achieve standard unforgeability against a single verification query, Proposition 3.6.1 and Theorem 3.4.2 imply that it MA-SUF secure against multiple verification queries.

PRF-based MACs. Any PRF-based MAC is MA-UF secure against 1 verification query [10], and, being a MAC, we can then apply the above to get MA-SUF security against multiple queries. It is worth stating the concrete security of the reduction underlying this easy result, however, since it provides the backdrop for our later improvement:

**Proposition 3.6.2** [PRF-based MACs are MA-SUF secure against multiple queries, basic reduction] Let $F : K \times M \to \{0, 1\}^\tau$ be a PRF and let $\Pi[F]$ be an associated MA scheme. Let $A$ be a $(t, q_s, q_v)$–adversary against $\Pi[F]$. Then there exists a PRF adversary $B$ who runs in time $T(t, q_s, q_v)$ and makes at most $q_s + 1$ PRF queries such that

$$\text{Adv}_{\Pi[F]}^{\text{MA-SUF}}(A) \leq \frac{q_v}{2^\tau} + q_v \cdot \text{Adv}_F^{\text{PRF}}(B).$$

(3.14)
Figure 3.6 Algorithms which constitute Carter-Wegman message authentication scheme $\Pi[H, F]$. Here $N$ is a $k$-bit counter which is initially 0 and then incremented with each invocation of SGN.

As an application, various CBC-MACs [10, 19, 53], PMAC [18], TMAC [45] and OMAC [39] are MA-SUF secure against multiple verification queries. (But Proposition 3.6.4 yields better results.)

**CARTER-WEГMAN MA SCHEMES.** The Carter-Wegman paradigm [62] is now yielding the most efficient MA schemes. UMAC [16, 17] is the canonical example. These schemes are either stateful or randomized and in particular are not MACs in the sense defined above, so the above does not apply. Via a closer look at the proof of the MA-UF security [44, 57, 36, 16, 17], however, one can establish MA-SUF security as well. Details follow. For simplicity we consider only the stateful case, meaning the counter-based scheme.

Let $H : \mathbb{K} \times M \rightarrow \{0, 1\}^\tau$ and $F : \mathbb{K} \times \{0, 1\}^k \rightarrow \{0, 1\}^\tau$ be some functions. We associate to them the counter-based CW MA scheme $\Pi[H, F] = (SGN, VF)$ over key space $\mathbb{K} \times \mathbb{K}$, message space $M$, tag space $\{0, 1\}^k \times \{0, 1\}^\tau$ as described in Figure 3.6.

**Proposition 3.6.3** [Security of CW MA schemes] Let families $H, F$ be as above, both with output-length $\tau$. Assume $H$ is an $\varepsilon$-AXU family and $F$ is a PRF. Let $\Pi[H, F]$ be the associated CW MA scheme as above. Let $A$ be any $(t, q_s, 1)$-adversary against $\Pi[H, F]$, where $q_s \leq 2^k$. Then there exists a PRF adversary $B$ such that

$$\text{Adv}_{\Pi[H, F]}^{\text{MA-SUF}}(A) \leq 2^{-\tau} + \varepsilon + \text{Adv}_{F}^{\text{PRF}}(B).$$

(3.15)

Furthermore, the number of oracle queries made by $B$ is $q_s + 1$ and the running time of $B$ is that of $A$ plus $O(q_s \tau)$. 
**Proof:** The main part of the security proof considers the thought experiment in which $F(K, \cdot)$ is replaced by a random function $f : \{0, 1\}^k \rightarrow \{0, 1\}^\tau$. Formally, this means we consider the security of the MA scheme $\Pi[H, R]$ where $R$ is the family of all functions having the same domain and output-length as $F$. We now bound $\text{Adv}_{\Pi[H,R]}^{\text{MA-SUF}}(A)$.

By assumption $A$ makes at most $2^k$ tagging queries, thus the tagging oracle never replies to two queries with the same nonce. Let $M, (N, S)$ denote the single verification query made by $A$. If the nonce $N$ is new, then clearly the probability that the tag is correct is at most $2^{-\tau}$ since $f$ has never before been invoked on input $N$.

However, there could have been a previous query $M'$ to the tagging oracle returning a tag of the form $(N, S')$. If $M' = M$ then the only way that $(N, S)$ is a correct tag for $M$ is that $S' = S$, so in this case there is no strong forgery. So assume $M' \neq M$. In that case the argument reduces to that of the MA-UF case. Namely, we have

$$S \oplus S' = (f(N) \oplus h(M)) \oplus (f(N) \oplus h(M')) = h(M) \oplus h(M')$$

and the probability that this can happen is at most $\varepsilon$ by assumption that $H$ is $\varepsilon$-AXU. In summary

$$\text{Adv}_{\Pi[H,R]}^{\text{MA-SUF}}(A) \leq 2^{-\tau} + \varepsilon.$$ 

Now one can lift this to the statement of the Proposition, using the assumption that $F$ is a PRF, in a standard way.

Since UMAC [16, 17] and MMH-MAC [36] were shown to be MA-UF secure against a single verification query in the cited papers, the above together with Theorem 3.4.2 shows they are MA-SUF secure against multiple verification queries.

### 3.6.2 Improvements for PRF Based MACs.

Although Corollary 3.5.2 precludes improved concrete security results for data authentication primitives in general, we note that one can improve the reduction for PRF-based MACs given in Proposition 3.6.2 to show the following:
Proposition 3.6.4 [PRF-based MACs are MA-SUF secure against multiple queries, improved version] Let $F : K \times M \to \{0,1\}^\tau$ be a PRF and let $\Pi[F]$ be an associated MA scheme. Let $A$ be a $(t,q_s,q_v)$-adversary against $\Pi[F]$. Then there exists a PRF adversary $B$ who runs in time $t + O((q_s + q_s)\tau)$ and makes at most $q_s + q_v$ PRF queries such that

$$
\text{Adv}_{\Pi[F]}^{\text{MA-SUF}}(A) \leq \frac{q_v}{2^\tau} + \text{Adv}_F^{\text{PRF}}(B). \quad (3.16)
$$

Proof: Adversary $B$ is given access to oracle $f: M \to \{0,1\}^\tau$. It runs $A$ and handles $A$’s queries as follows. When $A$ makes a query $M$ to its tagging oracle, $B$ responds with $f(M)$. When $A$ makes a query $(M, T)$ to its verification oracle, $B$ returns 1 to $A$ if $f(M) = T$ and 0 otherwise. Finally, $B$ outputs 1 if $A$ succeeded in strong forgery, and 0 otherwise. (In the first case it is betting $f$ is an instance of $F$, and in the second it is betting $f$ was chosen at random from Maps$(M, \{0,1\}^{\tau})$. The analysis is standard.]

The improvement relative to Proposition 3.6.2 is to eliminate the factor of $q_v$ multiplying $\text{Adv}_F^{\text{PRF}}(B)$ at the cost of allowing $B$ an extra $q_v$ PRF queries. However, typically $q_s \geq q_v$, so the number of oracle queries of $B$ in fact only increases by a factor of 2 and the improvement is essentially for free. Now, under typical choices of $\tau$, the $q_v2^{-\tau}$ term is negligible compared to the other term, so roughly the bound is better than that of Proposition 3.6.2 by a factor $q_v$. This forms the basis for the examples, discussed in Section 1.1.6, which showed that the improvement has appreciable practical impact.

We note that as applications we obtain MA-SUF security against multiple queries for the CBC-MACs [10, 19, 53], PMAC [18], TMAC [45] and OMAC [39] with concrete security that is better than that envisaged at the time people believed that MA-UF security against 1 query implies MA-UF security against multiple queries, even though we now know this implication is not even true.
Algorithm $\text{SGN}((h, f), M)$:

Pick distinct random points $r_1, \ldots, r_t \in \{0, 1\}^n$

$z \leftarrow h(M) \oplus f(r_1) \oplus \ldots \oplus f(r_t)$

Return $(r_1, \ldots, r_t, z)$

Algorithm $\text{VF}((h, f), M, (r_1, \ldots, r_t, z))$:

$z' \leftarrow h(M) \oplus f(r_1) \oplus \ldots \oplus f(r_t)$

If $z = z'$ then return 1

else return 0

---

**3.6.3 Generalized Carter-Wegman MACs**

Generalized Carter-Wegman MACs [7] are stateless MAC schemes which have concrete security well beyond the birthday bound. Generalized CW MACs were claimed in [7] to have MA-SUF security against multiple verification queries but as we discussed in Section 2.4, the security proof of [7] lacks some important details. Here we show how Theorem 3.4.5 can be used to prove the result which was claimed in [8]. We first describe the generalized CW MA scheme $\text{MACRX}_t[F]$ and then give the security result.

Let $H$ be a family of $\varepsilon$-AXU functions, in which each function $h \in H$ maps from $\{0, 1\}^* \to \{0, 1\}^n$, let $F$ be any set of functions with domain $\{0, 1\}^n$ and range $\{0, 1\}^m$ and let $t \geq 1$ be any integer. Generalized Carter-Wegman MAC $\text{MACRX}_t[F] = (\text{SGN}, \text{VF})$ associated to $H$, $F$ and $t$, as defined by Bellare, Goldreich and Krawczyk [7] is depicted in Figure 3.7. Keys for $\text{MACRX}_t[F]$ are tuples of functions $(h, f) \in H \times F$, messages are arbitrary binary strings and tags are $(t + 1)$-tuples from $\{0, 1\}^n \times \ldots \times \{0, 1\}^n \times \{0, 1\}^m$, which can be viewed as $tn + m$-bit strings.

The principle behind generalized CW MACs closely resembles that of standard
Carter-Wegman MACs which we discussed in Section 3.6.1. Informally, the authentication tag is a XOR of the values of a PRF on a small number of distinct nonces and the value of $\varepsilon$-AXU hash on an input message. Generalized CW MACs use multiple nonces while standard CW MACs use a single nonce. We remark that despite this similarity in constructions, the security proof for generalized CW MACs is significantly more involved. Intuitively, standard CW MACs can be shown secure if all nonces obtained by the adversary are distinct. For generalized CW MACs this condition (that all tuples of nonces are distinct) is simply not sufficient. There exist scenarios where the adversary obtains distinct tuples of nonces and still is in the position to do a successful forgery.

As we showed in Section 3.2, MA scheme $\text{MACRX}_t[F]$ itself is a data authentication primitive over sets $K = H \times F$, $M = \{0, 1\}^*$, $T = \{0, 1\}^{m+n}$ and $V = M \times T$ with freshness function $\text{Fresh}_{\text{MA-SUF}}$ and revealing function $\text{Reveal}_{\text{MA-SUF}}$. Any adversary against strong unforgeability of the MA scheme is also an adversary against the data authentication primitive with the same advantage.

Consider any $t \geq 1$ and any set $F$ of functions from $\{0, 1\}^n$ to $\{0, 1\}^m$. Consider the UF game with $\text{MACRX}_t[F]$ against adversaries who make no more than $q_s$ signing queries. Random choices made by the UF game include random choice of the key $K \in K$ and random choice of sequences $R_1, \ldots, R_{q_s}$ used in the signing queries, where each $R_i = (r_{i,1}, \ldots, r_{i,t})$ is a sequence of $t$ distinct $n$-bit strings.

To each sequence $R_i$ we associate its characteristic vector of length $N = 2^n$, denoted $ChV(R_i)$. If we consider the values $r_{i,1}, \ldots, r_{i,t}$ as representing integer numbers from 0 to $N - 1$ then $ChV(R_i)$ will have 1-s in the positions corresponding to these numbers and 0-s elsewhere. We denote by $MTX(R_1, \ldots, R_{q_s})$ the $q_s$ by $N$ matrix of zeros and ones whose $i$-th row is $ChV(R_i)$ for $i \in \{1, \ldots, q_s\}$.

Each set of random choices for the UF game defines matrix $MTX(R_1, \ldots, R_{q_s})$. The matrix is called $t$-vulnerable if either it has two identical rows or there is a subset $S \subset \{1, \ldots, q_s\}$ such that $\bigoplus_{i \in S} ChV(R_i) = v$, where $v \not\in S$ and contains exactly $t$ 1-entries. Let $\text{Hard}$ be the function on random coins of the UF game, which returns 0 if that matrix $MTX(R_1, \ldots, R_{q_s})$ is $t$-vulnerable and 1 otherwise; let $Ev_{\text{Hard}}$ denote the
event that $\text{Hard}$ returns 1.

Denote by $VulProb$ the probability that $MTX(R_1, \ldots, R_{q_v})$ is not $t$-vulnerable, where $R_1, \ldots, R_{q_v}$ are random sequences of distinct $n$-bit strings. Clearly,

$$\Pr[\neg Ev_{\text{Hard}}] = VulProb.$$ 

Bellare et al. [8] proved that

$$VulProb \leq d(n, t) \cdot \frac{q_s^3}{2^n}$$

for a certain function $d(n, t)$, which does not depend on $q_s$.

Let $F$ be a set of all functions from $\{0, 1\}^n$ to $\{0, 1\}^m$. Bellare, Goldreich and Krawczyk [8] proved that for any adversary $A$ against MA scheme $MACRX_t[F]$ who makes $q_s$ signing queries and 1 verification query, the probability that $A$ wins the unforgeability game assuming that the corresponding matrix $MTX(R_1, \ldots, R_{q_v})$ is not $t$-vulnerable, is less or equal to $\varepsilon$. This implies that for any $(\infty, q_s, q_v)$-adversary $A$ against data authentication primitive $MACRX_t[F]$ with freshness function $\text{Fresh}_{\text{MA-SUF}}$

$$\Pr[A_{UF} \text{ sets win} \mid Ev_{\text{Hard}}] \leq \varepsilon.$$ 

Applying Theorem 3.4.5 and translating the obtained results about a data authentication primitive to results about a MA scheme, we obtain the following corollary.

**Corollary 3.6.5** Let $H$ be any family of $\varepsilon$-AXU functions from $\{0, 1\}^* \to \{0, 1\}^m$, let $F$ be a set of all functions from $\{0, 1\}^n$ to $\{0, 1\}^m$ and let $t \geq 1$ be any integer. Then for any adversary $A$ against strong unforgeability of MA scheme $MACRX_t[F]$, who makes at most $q_s$ signing and $q_v$ verification queries,

$$\text{Adv}^\text{SUF}_{MACRX_t[F]}(A) \leq q_v \cdot \varepsilon + VulProb,$$

where

$$VulProb \leq d(n, t) \cdot \frac{q_s^3}{2^n}$$

for some function $d(n, t)$, which does not depend on $q_s$. $\blacksquare$

### 3.6.4 EAX Mode of Operation

EAX is a mode of operation of a block-cipher by Bellare, Rogaway and Wagner [15], which provides authenticated encryption with associated data (AEAD).
For any block cipher $E: K \times \{0,1\}^n \rightarrow \{0,1\}^n$ and for any tag length $\tau \in \{0,\ldots,n\}$ EAX provides an AEAD scheme $EAX[E, \tau] = (Enc, Dec)$ with a key space $K$, nonce space $N = \{0,1\}^*$, message space $M = \{0,1\}^*$, header space $H = \{0,1\}^*$ and ciphertext space $C = \{0,1\}^*$. EAX uses counter mode encryption algorithm CTR and a pseudorandom function OMAC [39]. The encryption algorithm of EAX is illustrated in Figure 3.8.

Analysis of the EAX mode involves specifying an extra adversarial resource. For any adversary against the EAX, its data complexity $\sigma$ is the maximal total length (in $n$-bit blocks) of all encryption queries made by the adversary plus the total number of these queries. As a standard argument, in security analysis of the EAX mode, actual block cipher is replaced a random function with domain $\{0,1\}^n$ and range $\{0,1\}^n$. Such a variant of the EAX mode is denoted by $EAX[\mathcal{R}_n^\tau]$.

Bellare et al. [15] reduced authenticity of the EAX mode to security of XOR-tag schemes. The following proposition can be inferred from [15]. In what follows $\Pi[m, \tau]$ denotes a XOR-tag scheme with parameters $m$ and $\tau$ (as defined in Section 3.2) and XOR-TAG denotes the original security experiment for XOR-tags [15].

**Proposition 3.6.6** [15] For any $(t, q_s, q_v)$-adversary $A$ against authenticity of AEAD scheme $EAX[\mathcal{R}_n^\tau]$, who has data complexity $\sigma$, the advantage of $A$ can be bounded by

$$\text{Adv}^\text{auth}_{EAX[E, \tau]}(A) \leq \frac{10.5\sigma^2}{2^n} + \delta(q_v),$$

(3.17)

where $\delta(q_v)$ is the maximal advantage of adversaries who make at most $q_v$ verification queries in XOR-TAG experiment against $\Pi[m, \tau]$ (for some $m \geq 1$).

Also, they established security of XOR-tag schemes against adversaries who make a single verification query and proved that for any adversary $B$ against security of XOR-tag scheme $\Pi[m, \tau]$ who makes at most 1 verification query,

$$\text{Adv}^\text{XOR-TAG}_{\Pi[m, \tau]}(B) \leq 1/2^\tau.$$

Since XOR-tag schemes are data authentication primitives which have a revealing function (see Section 3.2), Theorem 3.4.2 allows to prove that XOR-tag schemes are
secure against multiple verification queries. Translating this to the “language” of XOR-tag schemes we obtain the following.

**Proposition 3.6.7** For any adversary $A$ against XOR-tag scheme $\Pi[m, \tau]$, who makes at most $q_v$ verification queries,

$$\text{Adv}^{\text{XOR-TAG}}_{\Pi[m, \tau]}(A) \leq \frac{q_v}{2^\tau}.$$

Proposition 3.6.7 is an interesting result on its own and also it allows to prove a concrete security bound for the EAX mode against multiple-queries adversaries. Combining Proposition 3.6.7 with Proposition 3.6.6, we obtain the following result on the security of the EAX mode against adversaries making multiple verification queries.

**Corollary 3.6.8** For any $(t, q_s, q_v)$–adversary $A$ against authenticity of $EAX[R^n, \tau]$ with data complexity $\sigma$, the advantage of $A$

$$\text{Adv}^{\text{auth}}_{EAX[R^n, \tau]}(A) \leq \frac{10.5\sigma^2}{2^\tau} + \frac{q_v}{2^\tau}.$$

### 3.7 Acknowledgements

Parts of this chapter appeared in the paper


The dissertation author was the primary investigator and author of this paper.
Figure 3.8 Encryption under EAX. The message is $M$, the key is $K$, and the header is $H$. The ciphertext is $CT = C\|T$. 
4 Append-Only Signatures

4.1 Definitions

4.1.1 Append-only Signatures

Informally, append-only signatures (AOS) are signatures that enable the public extension of existing signatures. That is, any party given an AOS signature $\text{Sig}$ on a message $(M_1, \ldots, M_n)$ can compute an AOS signature on any message $(M_1, \ldots, M_n, M_{n+1})$. One could represent the message $(M_1, \ldots, M_n)$ as the string $M_1||\ldots||M_n$, which better captures the idea of appending. However, since we want to differentiate between a message of the form “A”||“B” and that of the form “AB” (“A”, “B” and “AB” being three different message symbols), we prefer to think of messages as $n$-tuples. Besides the append operation, AOS is the same as ordinary signatures. That is, given only an AOS signature on the message $(M_1, \ldots, M_n)$ it should be infeasible to forge an AOS signature on any message not having $(M_1, \ldots, M_n)$ as a prefix. In terms of conventional signatures, AOS may seem strange, as it allows the “forgery” of signatures on messages not previously obtained. In particular, given a signature on the empty message $\varepsilon$, a signature on any message $(M_1, \ldots, M_n)$ can be computed. In the context of AOS, we view this as a feature, and, as we will show, this is useful in several applications.

We now formally define AOS and the corresponding notion of security. Let AOS. MSpace be any set of symbols (for example, $\{0, 1\}$ or $\{0, 1\}^*$). For an integer $n \geq 0$, a message of length $n$ is an $n$ tuple of symbols written as $M[1..n] = (M_1, M_2, \ldots, M_n)$.
with \( M_i \in \text{AOS.MSpace} \). The special case of \( n = 0 \) is the empty message, denoted as \( \varepsilon \), also written as \( M[1..0] \). We use the symbol \( \sqsubseteq \) to denote the prefix relation over the messages: for a given message \( M[1..n] = (M_1, M_2, \ldots, M_n) \), any message from the set \( \{M[1..i], 0 \leq i \leq n\} \) is a prefix. Note that \( \varepsilon \) is a prefix of any other message.

An append-only signature (AOS) scheme w.r.t. the message space \( \text{AOS.MSpace} \) is a collection of three algorithms: a setup algorithm (AOS.Setup), an append algorithm (AOS.Append), and a verify algorithm (AOS.Vfy), defined as follows:

- AOS.Setup (the key generation algorithm) takes the security parameter as input and outputs a pair of keys: the public key AOS.pk and the secret key \( \text{Sig}[\varepsilon] \), which is the signature on the empty message \( \varepsilon \).

- AOS.Append (the append algorithm) takes the public key AOS.pk, a signature on a message \( M[1..n-1] = (M_1, \ldots, M_{n-1}) \), of length \( n-1 \), and a symbol \( M_n \in \text{AOS.MSpace} \) and produces a signature on the message \( M[1..n] = (M_1, \ldots, M_n) \).

- AOS.Vfy (the verification algorithm) takes the public key AOS.pk, a message \( M[1..n] \), and a signature \( \text{sig} \), and returns either \text{true} or \text{false}.

All algorithms can be randomized and all of them must be polynomial-time in the security parameter. Additionally, the scheme should have the property that for any pair \( (\text{AOS.pk}, \text{Sig}[\varepsilon]) \) generated by AOS.Setup(1\(^k\)) and any message \( M[1..n] = (M_1, M_2, \ldots, M_n) \) (where \( n \) is polynomially bounded in the security parameter), the signature on \( M[1..n] \) given by

\[
\text{sig} = \text{AOS.Append}(\text{AOS.pk}, M[1..n-1], \text{AOS.Append}(\text{AOS.pk}, M[1..n-2], \ldots, \ldots, \text{AOS.Append}(\text{AOS.pk}, \text{Sig}[\varepsilon], M_1), \ldots, M_{n-1}), M_n) \quad (4.1)
\]

should be accepted by a verification algorithm. That is, AOS.Vfy(\( \text{AOS.pk}, M[1..n], \text{sig} \)) must return \text{true}.

The way an AOS signature is defined in Eq. (4.4.1) implies that the only way of appending a sequence of symbols to a given AOS signature is to append the symbols
one-by-one. This means that the distribution of an AOS signature created by appending a symbol \( M_n \) to a message \((M_1, \cdots, M_{n-1})\) is the same as the distribution of the signature on the message \((M_1, \cdots, M_{n-1}, M_n)\) when generated from scratch (using the secret key \( \text{Sig}[\varepsilon] \)). This fact ensures history independence of AOS: that is, no party, given an AOS signature, can tell whether the signature was created by the owner of the secret key or whether it passed through multiple parties that appended symbols at every step\(^1\). History independence is a useful property to have in most applications (as already highlighted in previous work on algebraic signatures \([41]\) and incremental signatures \([6]\)).

**Definition 4.1.1** \([\text{AOS-UF-CMA}]\) Let \( \mathcal{AOS} = (\text{AOS.Setup}, \text{AOS.Append}, \text{AOS.Vfy}) \) be an AOS scheme, let \( k \) be the security parameter, and let \( \mathcal{A} \) be an adversary. We consider the experiment:

**Experiment** \( \text{Exp}_{\mathcal{AOS, A}}^{\text{aos-uf-cma}}(k) \)

\[
\begin{align*}
\text{MSGSet} & \leftarrow \emptyset; (\text{AOS.pk}, \text{Sig}[\varepsilon]) \leftarrow \text{AOS.Setup}(1^k) \\
(M[1..n], \text{sig}) & \leftarrow \mathcal{A}^{\text{AOSSign}(\cdot)}(\text{AOS.pk}) \\
\text{if } \text{AOS.Vfy}(\text{AOS.pk}, M[1..n], \text{sig}) = \text{true} \quad &\quad \text{and } \forall J[1..j] \subseteq M[1..n] : J[1..j] \not\in \text{MSGSet} \\
&\quad \text{then return } 1 \text{ else return } 0
\end{align*}
\]

**Oracle** \( \text{AOSSign}(M[1..n]) \)

\[
\text{MSGSet} \leftarrow \text{MSGSet} \cup \{M[1..n]\}
\]

**return** \( \text{EXTRACT}(M[1..n]) \)

**Oracle** \( \text{EXTRACT}(M[1..i]) \) // defined recursively

---

\(^1\)The above definition precludes trivial schemes of the following form: Let \( \mathcal{Sgn} = (\text{SGN.G}, \text{SGN.S}, \text{SGN.V}) \) be any standard digital signature scheme. Construct an append-only signature scheme using \( \mathcal{Sgn} \) in which the signature of any message \( M[1..n] = (M_1, \cdots, M_n) \) is \( \text{SGN.S}(M[1..n], n) \) and the program AOS.Append takes a message \( M[1..n] \), its signature \( (\sigma, n) \) and a new symbol \( M[n+1] \) and simply outputs \( (\sigma, n) \). Verification of a signature \( (\sigma, n) \) on message \( M[1..N] \) \((N \geq n)\) is carried out by testing if \( \sigma \) is the signature, according to \( \mathcal{Sgn} \), on \( M[1..n] \). Although this scheme allows appending to already signed messages in an arbitrary manner, one can easily distinguish between signatures created by such append operations and those created from scratch.
if \( i = 0 \) then return \( \text{Sig}\[\varepsilon] \)
else if \( \text{Sig}[M[1..i]] \) = defined
    then return \( \text{Sig}[M[1..i]] \)
else \( \text{Sig}[M[1..i]] \) $\leftarrow$ AOS.Append(AOS.pk, \( M[1..i-1] \), EXTRACT(\( M[1..i-1] \)), \( M_i \))
return \( \text{Sig}[M[1..i]] \)

The aos-uf-cma-advantage of an adversary \( \mathcal{A} \) in breaking the security of the scheme \( \text{AOS} \) is defined as

\[
\text{Adv}_{\text{AOS},\mathcal{A}}^{\text{aos-uf-cma}}(k) = \Pr \left[ \text{Exp}_{\text{AOS},\mathcal{A}}^{\text{aos-uf-cma}}(M[1..i]) = 1 \right],
\]

and \( \text{AOS} \) is said to be unforgeable under chosen message attacks (aos-uf-cma-secure) if the above advantage is a negligible function in \( k \) for all polynomial-time adversaries \( \mathcal{A} \).

Note that in our definition of security, adversary \( \mathcal{A} \) is given access to oracle \( \text{AOSSIGN} \) but not to oracle \( \text{EXTRACT} \). The latter is used internally by \( \text{AOSSIGN} \) to create intermediate signatures. The history independence property of AOS ensures that the adversary can get no advantage when given the power to decide “how” the signature on any message is to be created by \( \text{AOSSIGN} \) (for example, whether it asks for a message \((M_1, M_2)\) to be signed from scratch or by first signing \( M_1 \) and then appending \( M_2 \), it would get the same reply in return).

### 4.1.2 Public Key Signature Schemes

In some of our constructions of append-only signatures we will utilize regular digital signature schemes. The corresponding security proofs for AOS will involve the security definition for a digital signature scheme. For completeness, here we provide the formal definition for a signature scheme, as formalized by Goldwasser, Micali and Rivest [34].

A public key signature scheme \( SGN = (SGN.G, SGN.S, SGN.V) \) is a collection of three algorithms: a key generation algorithm \( SGN.G \), a signing algorithm \( SGN.S \),
and a verification algorithm SGN.V. These algorithms must be polynomial time in the security parameter and should have the following input/output specification:

- SGN.G takes as input the security parameter $1^k$ and outputs a secret key/public key pair $(sk, pk)$. The public key also includes some system parameters like the description of the message space SGN.MSpace.

- SGN.S takes as input a message $M \in$ SGN.MSpace and produces a string $sig$ which is called a signature of a message $M$.

- SGN.V takes as input a public key $pk$, a message $M$ and a signature $sig$ and returns either true or false.

The verification algorithm must accept all signatures produced by the signing algorithm, namely the following should hold for every $(sk, pk)$ produced by SGN.G($k$), every message $M \in$ SGN.MSpace and every choice of random coins:

$$SGN.V_{pk}(M, SGN.S_{sk}(M)) = true.$$ 

Next we define unforgeability under chosen message attacks (sig-uf-cma) for a signature scheme.

**Definition 4.1.2 [SIG-UF-CMA]** Let $SGN$ be a signature scheme, let $k$ be a security parameter, and let $A$ be an adversary. We consider the following experiment:

<table>
<thead>
<tr>
<th>Experiment $Exp^{sig-uf-cma}_{SGN,A}(1^k)$</th>
<th>Oracle $SGN(M)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$MSGSet \leftarrow \emptyset$;</td>
<td>$sig \leftarrow SGN.S_{sk}(M)$</td>
</tr>
<tr>
<td>$(sk, pk) \leftarrow SGN.G(1^k)$</td>
<td>$MSGSet \leftarrow MSGSet \cup {M}$</td>
</tr>
<tr>
<td>$(M, sig) \leftarrow A^{SIGN(\cdot)}(pk)$</td>
<td>return $sig$</td>
</tr>
<tr>
<td>if $SGN.V(M, sig) = true$ \quad and $M$ not in $MSGSet$, return 1</td>
<td></td>
</tr>
<tr>
<td>else return 0</td>
<td></td>
</tr>
</tbody>
</table>

The sig-uf-cma advantage of an adversary $A$ in breaking security of the scheme $SGN$ is defined as

$$Adv^{sig-uf-cma}_{SGN,A}(k) = Pr[Exp^{sig-uf-cma}_{SGN,A}(k) = 1].$$
Signature scheme $SG\mathcal{N}$ is said to be unforgeable under chosen message attacks (sig-uf-cma) if the above advantage is negligible function in $k$ for all polynomial-time adversaries $A$.

4.2 Efficient AOS Constructions

4.2.1 Certificate-Based Append-Only Signatures

We present an efficient construction of a provably-secure AOS scheme based on a public-key signature scheme. Let $SG\mathcal{N} = (SGN.G, SGN.S, SGN.V)$ be a signature scheme with a space of public keys $SGN.PKSpace$ and message space $SGN.MSpace = AOS.MSpace \times SGN.PKSpace$. (A formal definition of a public-key signature scheme including a security definition is given in Appendix 4.1.2.) That is, messages to be signed by $SG\mathcal{N}$ are tuples of the form $(M, pk)$, where $M \in AOS.MSpace$ and $pk \in SGN.PKSpace$. Intuitively, an AOS signature $\text{Sig}$ of a message $M[1..n]$ consists of the following elements:

$$(pk_1, sig_1, \ldots, pk_n, sig_n, sk_n),$$

where for $1 \leq i \leq n$, $(pk_i, sk_i)$ are random public/secret key pairs of the public-key signature scheme $SG\mathcal{N}$ and $sig_i$ is a signature on the tuple $(M_i, pk_i)$ under the secret key $sk_{i-1}$. Note that the secret key $sk_i$ used to sign $sig_i$ is entangled with $sig_{i+1}$ by signing its corresponding public key $pk_i$, thereby certifying its validity. For this reason, this construction is sometimes referred to as a certificate chain. The signature $sig_0$ needs to be signed with the secret key $sk_0$, which we define to be the master secret key.

More formally, we construct the AOS scheme $AOS1$ with the message space $AOS.MSpace$ as specified below:

- **AOS.Setup($1^k$)** (the setup algorithm):
  - Run $SGN.G(1^k)$ to get a pair $(sk_0, pk_0)$. Set $AOS.pk \leftarrow pk_0$ and $\text{Sig}[\varepsilon] \leftarrow (sk_0)$.
  - Return $(AOS.pk, \text{Sig}[\varepsilon])$. 
• AOS.Append(AOS.pk, Sig[M[1..n]], M_{n+1}) (the append algorithm):
  Parse Sig as (pk_1, sig_1, ..., pk_n, sig_n, sk_n). If n = 0, then Sig[ε] consists of a
  single secret key sk_0. Run SGN.G(1^k) to generate a pair (sk_{n+1}, pk_{n+1}). Com-
  pute sig_{n+1} ← SGN.S_{sk_n}(M_{n+1}, pk_{n+1}). Return (pk_1, sig_1, ..., pk_n, sig_n, pk_{n+1},
  sig_{n+1}, sk_{n+1}).

• AOS.Vfy(AOS.pk, M[1..n], Sig) (the verification algorithm):
  Parse Sig as (pk_1, sig_1, ..., pk_n, sig_n, sk_n). If n = 0, then Sig = (sk_0). Set pk_0
to be the master public key AOS.pk. For i = 1..n − 1 verify that SGN.V(pk_{i-1},
  sig_i, (M_i, pk_i)) = true. If any of the verifications fail, return false. If all
  the verifications succeed, verify that (sk_n, pk_n) is a valid secret key/public key
  pair: pick any message M ∈ SGN.MSpace and compute sig ↝ SGN.S(sk_n, M).
  Return true if SGN.V(pk_n, sig, M) = true and false otherwise.

The length of a signature of AOS1 grows linearly with the number of symbols in a
message. The efficiency of AOS1 is summarized in Table 4.1. We prove aos-uf-cma
security of AOS1 provided that the original public-key signature scheme SGN, is sig-
uf-cma secure (as defined in Appendix 4.1.2).

Theorem 4.2.1 The AOS scheme AOS1 is aos-uf-cma secure assuming that the public-
key signature scheme SGN, is sig-uf-cma secure.

The full proof of Theorem 4.2.1 is in Section 4.2.2. Here we sketch the main ideas of
why this construction works. Intuitively, in order to break the aos-uf-cma security of
AOS1, an adversary has two choices between which we must distinguish. First, she
could try to forge a signature on a prefix of a message she already knows the signature
of. Since a valid AOS1 signature of this prefix (say, of length n') has to contain the
secret key sk_{n'} in cleartext, this would imply a full break of the security of the signature
scheme. Second, the adversary could try to forge an AOS signature on a message that is
different from all those with known signatures. To do so, the adversary could use exist-
ing public/secret key pairs, meaning she has to produce (for some i) a new signature on a
tuple (M_i, pk_i) under an unknown secret key and a different message M_i. Otherwise, the
Table 4.1  Efficiency of certificate-based AOS. Data is given for messages of length $n$.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Certificate-based AOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature length</td>
<td>$n \text{ SGN} \times$ signatures, $n \text{ SGN} \times$ public keys, $1 \text{ SGN} \times$ secret key</td>
</tr>
<tr>
<td>Setup time</td>
<td>$1 \times \text{SGN}.G(\cdot)$</td>
</tr>
<tr>
<td>Append time</td>
<td>$1 \times \text{SGN}.G(\cdot), 1 \times \text{SGN}.S(\cdot)$</td>
</tr>
<tr>
<td>Verify time</td>
<td>$(n + 1) \times \text{SGN}.V(\cdot), 1 \times \text{SGN}.S(\cdot)$</td>
</tr>
</tbody>
</table>

adversary breaks the certificate chain. That is, at some position $i$, the adversary creates a fresh secret-public key pair $(sk_i, pk_i)$ and uses $sk_i$ to create $\text{sig}_i$. However, $\text{sig}_{i-1}$ is a signature on the public key $pk_i$ and the symbol $M_{i-1}$ under the secret key $sk_{i-1}$. In order to use a new secret key $sk_i$ to create $\text{sig}_i$, the adversary has to forge a signature under the unknown secret key $sk_{i-1}$. This clearly contradicts the uf-cma security of the signature scheme.

### 4.2.2  Security Proof for Certificate-Based AOS (Theorem 4.2.1)

We show that for any adversary $A$ against $\text{AOS}_1$, there exists an adversary $B$ against $\text{SGN}$ running in about the same time as $A$ such that

$$\text{Adv}_{\text{AOS}_1,A}^{\text{aos-uf-cma}}(k) \leq s \cdot \text{Adv}_{\text{SGN},B}^{\text{sig-uf-cma}}(k).$$

The reduction factor $s$ is the upper bound on the number of messages $M$ for which $\text{Sig}[M]$ is defined by the $\text{Extract}(\cdot)$ oracle in the experiment $\text{Exp}_{\text{AOS}_1,A}^{\text{forge-cma}}(k)$. This bound should be known by $B$ before she runs the simulation of $A$ and the bound should hold for any choice of the public key, for any random coins of $A$ and for any oracle responses. The number $s$ could be also upper bounded by $q_e \cdot d$, where $q_e$ is the maximal number of $\text{AOSSign}(\cdot)$ queries made by $A$ and $d$ is the maximal length of the queries.

Now we proceed to the construction of adversary $B$ who attacks the unforgeability of the public key signature scheme $\text{SGN}$. In the sig-uf-cma experiment the challenger runs a key generation algorithm $\text{SGN}.G(1^k)$ to get a pair of keys $(sk, pk)$ and gives $pk$ as input to $B$ as well as an access to the $\text{SGN.S}_{sk}(\cdot)$ oracle.

During its execution $B$ will construct a set $T$ that at each moment of time will
contain all the messages $M$ for which $\text{Sig}[M]$ is already defined by $\text{Extract}(\cdot)$. This set plays the same role as in the original aos-uf-cma experiment. The only difference is that now, for each message $M[1..n] \in T$, we will keep not only a signature $\text{Sig}[M[1..n]]$ but also a pair of elements $(sk_{M[1..n]}, pk_{M[1..n]})$. The latter will correspond to the secret key/public key pair for $\text{SGN}$ generated by $\text{Extract}(\cdot)$ oracle when computing the signature for $M[1..n]$. The pseudocode for the adversary $B$ is given below:

Adversary $B(1^k, pk)$:
- Pick $guess \leftarrow [1,\ldots,s]$; $ctr \leftarrow 1; T \leftarrow \emptyset$
- if $guess = 1$ then $pk_e \leftarrow pk$; $sk_e \leftarrow \text{false}$
- else $(sk_e, pk_e) \leftarrow \text{SGN.G}(1^k)$
- Set $\text{AOS.pk} \leftarrow pk_e$ and define $\text{Sig}[e] \leftarrow \{sk_e\}$
- Add $e$ to $T$
- Run $A(1^k, \text{AOS.pk})$ and answer its $\text{AOSSign}(\cdot)$ queries as follows:

$\text{AOSSign}(M[1..n])$:
- $\text{Sig} \leftarrow \text{Extract}(M[1..n])$
- parse $\text{Sig}$ as $\{pk_{M[1]}, sig_1, \ldots, pk_{M[1..n]}, sig_n, sk_{M[1..n]}\}$
- if $sk_{M[1..n]} \neq \text{false}$ then return $\text{Sig}$ to $A$ else halt $B$; return Failure.

$\text{Extract}(M[1..n])$:// defined recursively
- if $M[1..n] \in T$ then return $\text{Sig}[M[1..n]]$
- else
  - $\text{Sig} \leftarrow \text{Extract}(M[1..n - 1])$
  - parse $\text{Sig}$ as $\{pk_{M[1]}, sig_1, \ldots, pk_{M[1..n-1]}, sig_{n-1}, sk_{M[1..n-1]}\}$
  - $ctr \leftarrow ctr + 1$
  - if $guess = ctr$ then
    - $M^+[1..n+] \leftarrow M[1..n]$
    - set $(sk_{M^+[1..n+]}, pk_{M^+[1..n+]}) \leftarrow (\text{false}, pk)$
  - else $(sk_{M[1..n]}, pk_{M[1..n]}) \leftarrow \text{SGN.G}(1^k)$
if $sk_{M[1..n-1]} \neq \text{false}$ then $sig_n \leftarrow \text{SGN}.S_{sk_{M[1..n-1]}}(m_n, pk_{M[1..n]})$

else get $sig_n$ by querying $(m_n, pk)$ to the $\text{SGN}.S_{sk}(\cdot)$ oracle.

Define $\text{Sig}[M[1..n]] = \{pk_{M[1]}, sig_1, \ldots, pk_{M[1..n]}, sig_n, sk_{M[1..n]}\}$

Add $M[1..n]$ to $T$

return $\text{Sig}[M[1..n]]$.

Eventually $A$ halts and outputs $(M^*[1..n], \text{Sig}^*)$, where

$\text{Sig}^* = \{pk_1^*, sig_1^*, \ldots, pk_n^*, sig_n^*, sk_n^*\}$.

Find the maximal index $n^* \in [0 \ldots n]$ such that

$M^*[1..n^*] \in T$ and s.t. $pk_i^* = pk_{M^*[1..i]}$ for all $i = 1 \ldots n^*$.

if $M^*[1..n^*] \neq M^+[1..n^+]$ then return Failure

else if $n^* = n$ then pick a pair $(m', pk')$ which was not queried to $\text{SGN}.S_{sk}(\cdot)$;

compute $sig' \leftarrow \text{SGN}.S_{sk^*_n}(m', pk')$; output $\{(m', pk'), sig'\}$

if $n^* < n$ then output $\{(m^*_{n^*+1}, pk^*_{n^*+1}), sig^*_{n^*+1}\}$.

At a high level, the adversary $B$ works as follows. Before running $A$, it randomly selects an integer $\text{guess} \leftarrow [1..s]$ which corresponds to an index of some message that will be queried to the $\text{EXTRACT}(\cdot)$ oracle. Note that all messages are selected by $A$, so $B$ does not know in advance the actual message $M^+[1..n^+]$. (It is defined only at the time when $\text{guess}$-th new message is queried to $\text{EXTRACT}(\cdot)$.) Next, $B$ runs $A$ and simulates the AOSSign$(\cdot)$ oracle. $B$ follows the protocol of the original oracle to compute signatures of all the messages that do not contain the guessed one as a prefix.

For the guessed message $M^+[1..n^+]$, the adversary $B$ sets $pk_{M^+[1..n^+]}$ to be equal to the input public key $pk$ and $sk_{M^+[1..n^+]} \leftarrow \text{false}$. Therefore, if $A$ queries the guessed message to the AOSSign oracle, $B$ declares Failure since she does not know the secret key $sk$ which corresponds to $pk$. However, $B$ still can correctly answer all the AOSSign queries that contain $M^+[1..n^+]$ as a prefix by using the $\text{SGN}.S_{sk}(\cdot)$ oracle that is given by the Exp$^{\text{sig-uf-cma}}$ experiment. Also, since $\text{EXTRACT}(\cdot)$ is recursive, any message is added to $T$ only after all its prefixes are already in $T$.

Let $A$ terminate and output a forgery of a message $M^*[1..n]$. $B$ assumes that
the forgery is valid and that her guessed message \( M^+[1..n^+] \) is equal to the prefix \( M^*[1..n^*] \) of the forged message. \( M^*[1..n^*] \) is the longest prefix of \( M^*[1..n] \) such that all the public keys \( pk_1^*, \ldots, pk_n^* \) from the signature \( \text{Sig}^* \) match the stored public keys \( pk_{M^*[1]}, \ldots, pk_{M^*[1..n^*]} \). In this case, \( \mathcal{B} \) can easily make a forgery for \( \mathcal{SGN} \): if \( n^+ < n^* \) then \( \text{sig}_{n^*+1}^* \) is a forgery for \( \mathcal{SGN} \), otherwise (if \( n^+ = n^* \)) the secret key \( sk_{n^*} \) should match the unknown secret key \( sk \).

Next we bound the advantage of \( \mathcal{B} \). There are several events in the experiment we must consider.

\[ \text{FAIL} : \mathcal{B} \text{ does output Failure.} \]

Without loss of generality, we can assume that if \( \mathcal{B} \) does not fail then \( \mathcal{A} \) always outputs a forgery \( \text{Sig}^* \) for some message \( M^*[1..n] \) and that no prefix of \( M^*[1..n] \) was queried to \( \text{AOSSIGN}(\cdot) \) (if this does not hold, \( \mathcal{A} \) automatically loses). The second event is defined as

\[ \text{FORGE} : \text{the forgery of } \mathcal{A} \text{ is valid,} \]

that is, \( \text{AOS.Vfy}(\text{AOS.pk}, \text{Sig}^*, M^*[1..n]) = 1 \). This event is defined only if \( \mathcal{B} \) does not fail. Finally, we consider random variables \( M^+[1..n^+] \) and \( M^*[1..n^*] \). The former random variable, \( M^+[1..n^+] \), corresponds to a message, whose signature was defined at the time when \( \mathcal{B} \) sets \( \text{guess} = \text{ctr} \). If no such query was made, we set \( M^+[1..n^+] \leftarrow \bot \). The latter random variable, \( M^*[1..n^*] \), corresponds to the “good” prefix of the forgery \( M^*[1..n] \) returned by \( \mathcal{A} \). It is defined only if \( \mathcal{B} \) does not fail. We define \( \text{GUESS} \) to be the event that \( \mathcal{B} \) guesses the message \( \mathcal{A} \) outputs a forgery on:

\[ \text{GUESS} : M^+[1..n^+] = M^*[1..n^*]. \]

We observe that if \( \mathcal{B} \) does not fail, \( \mathcal{A} \) wins and the guess is correct; \( \mathcal{B} \) then outputs a valid forgery of the signature scheme \( \mathcal{SGN} \). Therefore

\[
\text{Adv}^{\text{sig-uf-cma}}_{\mathcal{SGN}, \mathcal{B}}(k) \geq \Pr \left[ \neg \text{FAIL} \land \text{FORGE} \land \text{GUESS} \right]. \tag{4.2}
\]

The analysis is based on the following two claims. The first claim establishes that if \( \mathcal{B} \) guessed the right value for \( M^+[1..n^+] \) and \( \mathcal{B} \) does not output \text{failure} then the simulation of \( \mathcal{A} \) is perfect and we have
Claim 4.2.2 \( \text{Adv}^{\text{aos-uf-cma}}_{\text{AOS1},A}(k) = \Pr[\text{FORGE} | \neg\text{FAIL} \wedge \text{GUESS}] \).

The second claim shows that the probability of a correct guess is exactly \( \frac{1}{s} \):

Claim 4.2.3 \( \Pr[\neg\text{FAIL} \wedge \text{GUESS}] = \frac{1}{s} \).

We will settle the two claims later. Combining Claims 4.2.2 and 4.2.3 with Eqn. (4.2) we get

\[
\text{Adv}^{\text{sig-uf-cma}}_{s \in \mathcal{K},B}(k) \geq \Pr[\text{FORGE} \wedge \neg\text{FAIL} \wedge \text{GUESS}] \\
= \Pr[\text{FORGE} | \neg\text{FAIL} \wedge \text{GUESS}] \cdot \Pr[\neg\text{FAIL} \wedge \text{GUESS}] \\
= \frac{1}{s} \cdot \text{Adv}^{\text{aos-uf-cma}}_{\text{AOS1},A}(k),
\]

which completes the proof of the theorem.

Proof of Claim 4.2.2: We will show that in an AOS\text{SIGN}(\cdot) query \( M[1..n] \) made by \( A, B \) declares \text{Failure} if \( M[1..n] = M^+[1..n^+] \) and otherwise returns a signature which is distributed identically to the original aos-uf-cma experiment against \( \text{AOS1} \). This implies

\[
\Pr[\text{FORGE} | \neg\text{FAIL}] = \text{Adv}^{\text{aos-uf-cma}}_{\text{AOS1},A}(k).
\]

(4.3)

Since the output of \( A \) is independent from the choice of \textit{guess}, then the above equality holds also assuming \( M^+[1..n^+] = M^*[1..n^+] \):

\[
\Pr[\text{FORGE} | \neg\text{FAIL} \wedge \text{GUESS}] = \text{Adv}^{\text{aos-uf-cma}}_{\text{AOS1},A}(k).
\]

It is left to show (4.3): that if \( B \) does not return \text{Failure} then the input of \( A \) is identically distributed to the original aos-uf-cma experiment.

First, from the construction of \( B \) we see that the claim is true for messages that do not contain the guessed message \( M^+[1..n^+] \) as a prefix. In this case, the \text{EXTRACT} oracle uses the \text{AOS}.	ext{Append} algorithm of \( \text{AOS1} \) to append the signatures and therefore all signatures are distributed identically to the ones constructed by the \text{EXTRACT} oracle in the original aos-uf-cma experiment.
In the case when \( \mathcal{A} \) queries \( M[1..n] = M^+[1..n^+] \) to the \text{AOSSign}(\cdot) oracle, the \text{AOSSign}(\cdot) oracle calls \text{EXTRACT}(M^+[1..n^+]) to get \text{Sig}[M^+[1..n^+]]. The signature \text{Sig}[M^+[1..n^+]] returned by the \text{EXTRACT} oracle to has a form \( \{pk_1, sig_1, \ldots, pk_{n^+}, sig_{n^+}, sk_{n^+}\} \), where \( sk_{n^+} = \text{false} \) and \( \mathcal{B} \) declares \text{Failure} on such a query.

We are left to show what happens when \text{AOSSign} query \( M[1..n] \) contains \( M^+[1..n^+] \) as a prefix and \( n^+ < n \). We know that signatures of all the prefixes of length smaller that \( n^+ \) are correctly distributed. The recursive \text{EXTRACT} oracle constructs \text{Sig}[M[1..n^+]] and appends it using the \text{SGN.S_{sk}(\cdot)} oracle. The signature \text{Sig}[M[1..n^+]] has the form \( \{pk_1, sig_1, \ldots, pk_{n^+}, sig_{n^+}, sk_{n^+}\} \), where \( pk_{n^+} = pk \) and \( sk_{n^+} = \text{false} \). The input public key \( pk \) was generated by \text{SGN.G}(1^k) \) and thus \( pk_{n^+} \) is correctly distributed. To append \text{Sig}[M[1..n^+]] \) with \( M_{n^++1}, \text{EXTRACT}(\cdot) \) computes \( (sk_{n^++1}, pk_{n^++1}) \leftarrow \text{SGN.G}(1^k) \), \text{queries} \( sig_{sk}(M_{n^++1}, pk_{n^++1}) \) and sets \text{Sig}[M[1..n^++1]] \leftarrow \{pk_1, sig_1, \ldots, pk_{n^++1}, sig_{n^++1}, sk_{n^++1}\}. \) This signature is correctly distributed and all the following signatures are constructed by appending this one using \text{AOS.Append}. \]

\textbf{Proof of Claim 4.2.3:} Let \( \mathcal{A} \) terminate and output a forgery for \( M^*[1..n] \). Without loss of generality we can assume that no prefix of \( M^*[1..n] \) is contained in \text{MSGSet}. Thus the target message \( M^*[1..n^*] \) must be different from all the \text{AOSSign} queries made by \( \mathcal{A} \). The previous claim established that \( \mathcal{A} \)'s \text{AOSSign} queries are totally independent from the choice of \( M^+[1..n^+] \).

The index of the guessed message was uniformly chosen from \([1..s]\). If there exists an \text{AOSSign} query equal to \( M^+[1..n^+] \), \text{then} \( \mathcal{B} \) declares \text{Failure}; otherwise \( \mathcal{A} \) outputs message \( M^*[1..n^*] \) which must be different from all the \text{AOSSign} queries. Therefore the probability that \( \mathcal{B} \) does not fail and that \( M^*[1..n^*] = M^+[1..n^+] \) is exactly \( 1/s \).

\textbf{4.2.3 Shorter Signatures via Aggregation}

An aggregate signature, \( \mathcal{ASGN} = (\text{ASGN.G}, \text{ASGN.S}, \text{ASGN.AGG}, \text{ASGN.V}) \), allows the aggregation of \( n \) signatures on \( n \) distinct messages from \( n \) distinct users into a single signature. Its verification algorithm, \( \text{ASGN.V}(n, \cdot) \), takes an aggregated signature,
Table 4.2  Efficiency of AOS with signature aggregation. Data is given for messages of length \( n \).

<table>
<thead>
<tr>
<th>Metric</th>
<th>Certificate-based AOS with aggregation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature length</td>
<td>( 1 \text{ ASGN} ) signature, ( n \text{ ASGN} ) public keys, ( 1 \text{ ASGN} ) secret keys</td>
</tr>
<tr>
<td>Setup time</td>
<td>( 1 \times \text{ASGN.G}(\cdot) )</td>
</tr>
<tr>
<td>Append time</td>
<td>( 1 \times \text{ASGN.G}(\cdot), 1 \times \text{ASGN.S}(\cdot), 1 \times \text{ASGN.AGG}(\cdot) )</td>
</tr>
<tr>
<td>Verify time</td>
<td>( 1 \times \text{ASGN.V}(n, \cdot), 1 \times \text{ASGN.V}(1, \cdot), 1 \times \text{ASGN.S}(\cdot) )</td>
</tr>
</tbody>
</table>

\( n \) messages, and \( n \) public keys and verifies that the \( n \) users signed the \( n \) messages. A sequential signature aggregation algorithm assumes to receive the signatures sequentially: given an aggregated signature of \( n - 1 \) messages and a signature on an \( n^{th} \) message, it outputs an aggregated signature for all \( n \) messages.

When using the certificate-based construction of AOS from Section 4.2.1, we can use sequential signature aggregation to shrink the size of the signature (without significantly decreasing security or efficiency). To be more precise, the length of an AOS signature of a message of length \( n \) can be condensed to one signature of \( \text{ASGN} \), \( n \) public keys of \( \text{ASGN} \), and one secret key of \( \text{ASGN} \). We summarize the efficiency of this approach in Table 4.2. We note that there are two known signature aggregation techniques. The first scheme, given in [24], is based on bilinear maps. The second scheme (only supporting sequential aggregation) is from [48] and can be based on homomorphic trapdoor permutations (such as RSA). We note that both aggregation schemes are in the random oracle model.

### 4.2.4 Compact Signatures via the Boneh-Goh-Boyen HIBE

Even if we apply the techniques of signature aggregation to our certificate-based AOS scheme, the signature length remains linear in \( n \). Based on a recent HIBE construction from Boneh, Goh, and Boyen [21] we construct an AOS scheme \( AOS_2 \) whose signatures are of size square root of the maximum message length. The scheme is in fact a hybrid between the two schemes from [21, 22] exploiting a common algebraic structure. It is based on bilinear groups. The efficiency of this scheme is given in Table 4.3.
Table 4.3  Efficiency of \(AOS_2\). \(d\) represents the maximum message length. \(e(\cdot, \cdot)\) is a pairing operation on elements of the group \(G_1\) as used by the HIBE scheme.

<table>
<thead>
<tr>
<th>Metric</th>
<th>HIBE-based (AOS_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature length</td>
<td>(\leq 2\sqrt{d} + 1) elements (\in G_1)</td>
</tr>
<tr>
<td>Setup time</td>
<td>(1 \times e(\cdot, \cdot), 1) exp, (2\sqrt{d} + 1) rand number gen (\in G_1)</td>
</tr>
<tr>
<td>Append time</td>
<td>(\leq \sqrt{d} + 3) exp/mult</td>
</tr>
<tr>
<td>Verify time</td>
<td>((\sqrt{d} + 1) \times e(\cdot, \cdot), d + \sqrt{d}) exp/mult</td>
</tr>
</tbody>
</table>

Before describing the scheme, we review the hybrid HIBE scheme from Boneh, Goh and Boyen (see Section 4 of [21]) \((BGB - HIBE)\) achieving short private keys. By the results from Section 4.3 we know that a HIBE scheme implies an AOS scheme. Motivated by this reduction, we present a concrete AOS scheme (called \(AOS_2\)) based on \(BGB - HIBE\). In presenting the concrete scheme, we are able to make some (straightforward) efficiency improvements over the generic reduction by making the AOS verification algorithm deterministic (instead of probabilistic).

The main intention of this section is to demonstrate that AOS (and hence also HIBS) can be carried out with secret key size of order “square root of the length of the signed message”.

We briefly review the necessary facts about bilinear maps and bilinear groups. Let \(G_1\) and \(G_2\) be groups with the following properties.

- \(G_1\) is an additive group of prime order \(q\).
- \(G_1\) has generator \(P\).
- There is a bilinear map \(e : G_1 \times G_1 \rightarrow G_2\).

We have stipulated that our groups should have a bilinear map \(e : G_1 \times G_1 \rightarrow G_2\). This should satisfy the conditions below.

**Bilinear:** For all \(U, V \in G_1, a, b \in \mathbb{Z}\), \(e(aU, bV) = e(U, V)^{ab}\)

**Non-degenerate:** \(e(P, P) \neq 1_{G_2}\)
We say that $G_1$ is a *bilinear group* if there exists a group $G_2$ with $|G_2| = |G_1| = q$ and a bilinear map $e$ satisfying the conditions above; moreover, the group operations in $G_1$ and $G_2$ and $e$ must be efficiently computable.

Let $G_1$ be a bilinear group. We want to build an AOS scheme for messages of length at most $d$. For simplicity assume that $d$ has an integer root; that is, $l = \sqrt{d}$. Any message $M[1..n] = (M_1, \ldots, M_n)$ of length $n \leq d$ can be represented in matrix form as

$$M[1..n] = \begin{pmatrix} M_1 & M_2 & \ldots & M_l \\ M_{l+1} & M_{l+2} & \ldots & M_{2l} \\ \vdots & \vdots & \ddots & \vdots \\ M_{(l-1)l+1} & M_{(l-1)l+2} & \ldots & M_{l^2} \end{pmatrix} = \begin{pmatrix} M_{1,1} & M_{1,2} & \ldots & M_{1,l} \\ M_{2,1} & M_{2,2} & \ldots & M_{2,l} \\ \vdots & \vdots & \ddots & \vdots \\ M_{l,1} & M_{l,2} & \ldots & M_{l,l} \end{pmatrix}.$$  

We denote the induced index mapping as $\pi : \{1, \ldots, d\} \rightarrow \{1, \ldots, l\} \times \{1, \ldots, l\}$ with $\pi(n) = (n_1, n_2)$. Here $n_1$ is called the *row index* and $n_2$ the *column index*.

We may now describe $\text{AOS}_2 = (\text{AOS.Setup, AOS.Append, AOS.Vfy})$ giving its constituting algorithms.

- **AOS.Setup($1^k$):** Pick two random generators $P, Q \in G_1$ and a random element $\alpha \in \mathbb{Z}_q$. Next, pick random elements $G_1, \ldots, G_l$ and $H_1, \ldots, H_l \in G_1$. The public key consists of

  $$\text{AOS.pk} = (P, G_1, \ldots, G_l, H_1, \ldots, H_l, e(\alpha P, Q)).$$

  The root key $\text{Sig}[\varepsilon]$ is $\alpha Q$.

- **AOS.Append:** Let $(n_1, n_2) = \pi(n)$. For a message $M[1..n]$ the signature is defined as

  $$\text{Sig}[M[1..n]] = (\alpha Q + \sum_{i=1}^{n_1-1} r_i (G_i + \sum_{j=1}^l M_{i,j} H_j) + r'_{n_1} (G_{n_1} + \sum_{j=1}^{n_2} M_{n_1,j} H_j),$$

  $$r_1 P, \ldots, r_{n_1-1} P, r'_1 P, r'_{n_1} H_{n_2+1}, \ldots, r'_{n_1} H_l) \in G_1^{l+1+n_1-n_2},$$

  where $r_1, \ldots, r_{n_1-1}$, as well as $r'_{n_1}$ are random elements from $\mathbb{Z}_q$. 


We now describe the append operation. Given the public key AOS.pk, a signature \( \text{Sig}([M[1..n-1]]) \), and a new symbol \( M_n \), we want to output a valid signature of the message \( M[1..n] \).

For the append process we have to distinguish between two cases:

**Case 1:** \( \pi(n-1) = (n_1, n_2 - 1) \) (namely \( n \) and \( n - 1 \) have the same row index).

Then the signature on message \( M[1..n-1] \) has the form

\[
\left( \alpha Q + \sum_{i=1}^{n_1-1} r_i G_i + \sum_{j=1}^l M_{i,j} H_j \right) + r_n (G_{n_1} + \sum_{j=1}^{n_2-1} M_{n_1,j} H_j), \]

\[
r_1 P, \ldots, r_n P, r_{n_2} H_{n_2}, \ldots, r_{n_1} H_l \right) = (A, A_1, \ldots, A_{n_1}, B_{n_2}, \ldots, B_l).
\]

Pick a random \( r \in \mathbb{Z}_q \) and output the signature \( \text{Sig}[M[1..n]] \) as

\[
\left( A + M_{n_1,n_2} B_{n_2} + r (G_{n_1} + \sum_{j=1}^{n_2} M_{n_1,j} H_j), \right) \]

\[
A_1, \ldots, A_{n_1-1}, A_{n_1} + r P, B_{n_2+1} + r H_{n_2+1}, \ldots, B_l + r H_l \right).
\]

It is easy to verify that the resulting signature is of the right form with \( r'_{n_1} = r_{n_1} + r \).

**Case 2:** If \( \pi(n-1) = (n_1 - 1, 1) \) (the row indexes of \( n \) and \( n - 1 \) differ). Then the signature on message \( M[1..n-1] \) has the form

\[
\left( \alpha Q + \sum_{i=1}^{n_1} r_i G_i + \sum_{j=1}^l M_{i,j} H_j, r_1 P, \ldots, r_{n_1-1} P \right) = (A, A_1, \ldots, A_{n_1-1}).
\]

In this case we have \( \pi(n) = (n_1, 1) \) and to generate a signature for the message \( M[1..n] \) chose a random \( r \in \mathbb{Z}_q \) and compute \( \text{Sig}[M[1..n]] \) as

\[
\left( A + r (M_{n_1,1} H_1 + G_{n_1}), A_1, \ldots, A_{n_1-1}, r P, r H_2, \ldots, r H_l \right).
\]

- **AOS.Vfy.** Given a signature \( \text{Sig}[M[1..n]] \) and a message \( M[1..n] \), verification is
carried out as follows: Verify if

\[ e(\text{Sig}[M[1..n]], P) = \]
\[ e(\alpha P, Q) \cdot \prod_{i=1}^{n_1-1} (e(r_i P, G_i + \sum_{j=1}^{l} M_{i,j} H_j)) \cdot e(r'_n P, G_{n_1} + \sum_{j=1}^{n_2} M_{n_1,j} H_j). \]

The length of the signature of AOS2 grows linearly with the square root of the length of the message. Security is proved with respect to the \( l + 1 \)-Bilinear Diffie-Hellman Exponent (BDHE) assumption. (See [21] for an exact definition.)

**Theorem 4.2.4** Suppose the computational \( l + 1 \)-BHDE assumptions holds in the group \( \mathbb{G}_1 \). Then the AOS2 scheme is selective message aos-uf-cma secure.

Here selective message aos-uf-cma security is defined similar to Definition 4.1.1. The difference is that in the security experiment, the adversary has to commit to the message she is going to forge the signature for before the public/secret keys are issued and the public key is given to her. In the random oracle model, the scheme can be modified to get a full unforgeable AOS scheme. However, the security reduction is not tight and only allows for signatures on messages of constant length. Again, we refer the reader to [21] for more details.

The proof of the theorem follows that of the \( \mathcal{BG}B - \mathcal{HIBE} \) scheme from [21] and is omitted here. We note that since we are dealing with signatures instead of encryption, we can prove security of our scheme with respect to a computational assumption (rather than a decisional assumption as the original \( \mathcal{BG}B - \mathcal{HIBE} \) scheme).

### 4.2.5 AOS via Hash Trees

If the number of symbols in the alphabet \( \text{AOS.MSpace} \) is small, AOS can be efficiently implemented using hash trees [47]. This approach suffers from dramatic complexity blowup as the size of the message space increases, but uses only secret-key primitives and provides good security guarantees. We believe that this construction is useful in computationally constrained applications.
Next we construct an AOS scheme denoted by $\text{AOS}3$ with fixed message space $\text{AOS.MSpace} = \{0, 1\}$; the messages of $\text{AOS}3$ are limited to length at most $d$. The construction uses a pseudorandom generator and a collision-resistant hash function (for a formal definition of the two primitives we refer the reader to the textbook of Goldreich [31]). Let $G : \{0, 1\}^k \to \{0, 1\}^{2k}$ be a pseudorandom generator. Denote $G_i : \{0, 1\}^k \to \{0, 1\}^k$ to be the $i$-th $k$-bit component of $G$ for $i \in \{0, 1\}$. Let $H : \{0, 1\}^{2k} \to \{0, 1\}^k$ be a collision-resistant hash function.

Consider the left graph $T$ depicted in Figure 4.1. $T$ consists of the upper tree $UT$ and lower tree $LT$. The top node $\varepsilon$ is called the source and the bottom node $\bar{\varepsilon}$ is called the destination. Let $\langle v_1, \ldots, v_j \rangle$ denote the node at level $j$ below $\varepsilon$ (in the upper tree) such that each $v_i \in \{0, 1\}$ is an index of a node taken at the $i$-th level on the path from $\varepsilon$ to $\langle v_1, \ldots, v_j \rangle$. A mirror image of this node in the lower tree is denoted as $[v_1, \ldots, v_j]$.

Let $u = \langle v_1, \ldots, v_j \rangle$ be any node in the upper tree of the graph. We define the complement of $u$, denoted $\text{Comp}(u)$, to be the minimal set of nodes in $LT - \{\bar{\varepsilon}\}$ such that every path from $\varepsilon$ to $\bar{\varepsilon}$ passes through exactly one node from $\{u\} \cup \text{Comp}(u)$. An example of a complement set is given in Figure 4.1 (the right graph). Let $\neg$ denote the not operator. Then $\text{Comp}(u) = \{[v_1, \ldots, v_{i-1}, \neg v_i] \mid i = 1, \ldots, j\}$.

In $\text{AOS}3$, we associate every node of the graph $T$ with a secret key. Keys are assigned in a top-down manner, starting with a random key for the root $\varepsilon$ of $UT$. Furthermore, in $UT$ everyone can compute $\text{key}(\langle v_1, \ldots, v_j, 0 \rangle)$ and $\text{key}(\langle v_1, \ldots, v_j, 1 \rangle)$ from $\text{key}(\langle v_1, \ldots, v_j \rangle)$ using the pseudo random generators $G_0(\cdot), G_1(\cdot)$. Similarly, for nodes in $LT$ everybody can compute $\text{key}([v_1, \ldots, v_j])$ from $\text{key}([v_1, \ldots, v_j, 0])$ and $\text{key}([v_1, \ldots, v_j, 1])$ using the hash function $H(\cdot, \cdot)$. The nodes between $LT$ and $UT$ are “connected” through the pseudorandom generator $G_0(\cdot)$. The secret key is $\text{key}(\varepsilon)$, the public key is $\text{key}(\bar{\varepsilon})$. The AOS of a node $\langle v_1, \ldots, v_n \rangle$ (representing the message $M[1..n] = (M_1, \ldots, M_n) \in \{0, 1\}^n$, $n \leq d$) is given by the set of keys $\{\text{key}(x) \mid x \in \{\langle v_1, \ldots, v_n \rangle\} \cup \text{Comp}(\langle v_1, \ldots, v_n \rangle)\}$. Verification of a given AOS is done by computing top-down the corresponding keys in $LT$ and checking if the last key matches the public key $\text{key}(\bar{\varepsilon})$. The algorithms constituting $\text{AOS}3$ are defined as follows:
Figure 4.1 Structure of the hash-tree construction for $d = 2$. The diagram on the left depicts the hash tree. The diagram on the right highlights the node $u = (0, 1)$ (shown in black) and the set of its complements, $\text{Comp}(u)$ (shown in gray).

- **AOS.Setup($1^k$):**

  Pick $\text{key}(\varepsilon) \xleftarrow{\$} \{0, 1\}^k$; $\text{Sig}[\varepsilon] \leftarrow \text{key}(\varepsilon)$.

  Compute $\text{key}(u) \in \{0, 1\}^k$ for all nodes $u \in T$ as follows:

  For every node $\langle v_1, \ldots, v_j \rangle \in UT$ recursively compute
  
  \[
  \text{key}(\langle v_1, \ldots, v_j \rangle) = G_{v_j}(\text{key}(\langle v_1, \ldots, v_{j-1} \rangle)).
  \]

  For every node $c = [v_1, \ldots, v_d]$ at the $d$-th level of $LT$
  
  \[
  \text{key}(c) = G_0(\langle v_1, \ldots, v_j \rangle).
  \]

  For the remaining nodes in $LT$ recursively compute
  
  \[
  \text{key}([v_1, \ldots, v_j]) = H(\text{key}([v_1, \ldots, v_j, 0]), \text{key}([v_1, \ldots, v_j, 1]));
  \]

  The public key is $\text{key}(\tilde{\varepsilon}) = H(\text{key}([0]), \text{key}([1]))$.

  Return $(\text{key}(\varepsilon), \text{key}(\tilde{\varepsilon}))$. 
Table 4.4 Efficiency of the hash-tree based scheme AOS3. $d$ represents the maximum message length, $n$ represents the length of a given message, and $k$ is the security parameter.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Hash-tree based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature length</td>
<td>$(n + 1) \cdot k$-bit blocks</td>
</tr>
<tr>
<td>Setup time</td>
<td>$(2^{d+1} - 1) \times G(\cdot), (2^d - 1) \times H(\cdot, \cdot)$</td>
</tr>
<tr>
<td>Append time</td>
<td>$2^{d-n} \times G(\cdot), (2^{d-n} - 1) \times H(\cdot, \cdot)$</td>
</tr>
<tr>
<td>Verify time</td>
<td>$(2^{d-n+1} - 1) \times G(\cdot), (2^{d-n} + n - 1) \times H(\cdot, \cdot)$</td>
</tr>
</tbody>
</table>

- **AOS.Append**($\text{Sig}[M[1..n]], M_{n+1}$): //denote $\langle M_1, \ldots, M_n \rangle$ as $u$
  Parse $\text{Sig}$ as a set $\{\text{key}(x) | x \in \{u\} \cup \text{Comp}(u)\}$.
  Compute $\text{key}(\langle M_1, \ldots, M_{n+1} \rangle) = G_{M_{n+1}}(\text{key}(\langle M_1, \ldots, M_n \rangle))$.
  By iterating $G(\cdot)$ and $H(\cdot, \cdot)$
    compute $\text{key}([M_1, \ldots, M_n, \neg M_{n+1}])$ from $\text{key}(\langle M_1, \ldots, M_n \rangle)$
  // Note that $\text{Comp}(\langle M_1, \ldots, M_{n+1} \rangle) = [M_1, \ldots, M_n, \neg M_{n+1}] \cup \text{Comp}(u)$.
  Return $\{\text{key}(x) | x \in \{\langle M_1, \ldots, M_{n+1} \rangle\} \cup \text{Comp}(\langle M_1, \ldots, M_{n+1} \rangle)\}$.

- **AOS.Vfy**($\text{AOS.pk, M[1..n], Sig}$):
  Let $u = \langle M_1, \ldots, M_n \rangle$. Parse $\text{Sig}$ as a set $\{\text{key}(x) | x \in \{u\} \cup \text{Comp}(u)\}$.
  By iterating $G(\cdot)$ and $H(\cdot, \cdot)$
    compute $\text{key}(\cdot)$ for all the descendants of $\{u\} \cup \text{Comp}(u)$.
  If $\text{key}(\tilde{\varepsilon}) = \text{AOS.pk}$ return 1, otherwise return 0.

We give the efficiency of this scheme in Table 4.4. We prove aos-uf-cma security of the AOS3 scheme assuming the security of the underlying functions $G(\cdot)$ and $H(\cdot, \cdot)$:

**Theorem 4.2.5** If $G(\cdot)$ is a secure pseudorandom generator, $G_0(\cdot)$, $G_1(\cdot)$, are secure one-way functions and $H(\cdot, \cdot)$, $G_0(\cdot)$, and $G_1(\cdot)$ are all collision-resistant hash functions, then AOS3 is aos-uf-cma secure.
4.2.6 Security Proof for Hash Tree AOS (Theorem 4.2.5)

In this section we will establish the following: for any adversary \( A \) against aos-uf-cma security of AOS3, there exists an adversary \( B \) (against collision-resistance of \( G_0, G_1 \) and \( H \)), \( C \) (against one-wayness of \( G_0 \) and \( G_1 \)) and \( D \) (against pseudo-randomness of \( G \)) all running in about the same time as \( A \) such that

\[
\text{Adv}_{\text{AOS3}, A}^{\text{aos-uf-cma}}(k) \leq \sum_{S \in \{H, G_0, G_1\}} \text{Adv}_{S, B}^{\text{cr}}(k) + 3 \cdot 2^d \left( \sum_{S \in \{G_0, G_1\}} \text{Adv}_{S, C}^{\text{1-way}}(k) + 2d \cdot \text{Adv}_{G, D}^{\text{prg}}(k) \right).
\]

Here \( \text{Adv}_{S, B}^{\text{xxx}}(k) \) denotes the advantage of a (polynomially bounded) adversary \( B \) attacking the \( \text{xxx} \) security of the primitive \( S \), where \( \text{xxx} \) can be \( \text{cr} \) (collision resistance), \( \text{1-way} \) (one-wayness), or \( \text{prg} \) (pseudorandomness). For a formal definition we refer the reader to the textbooks of Goldreich [31, 32]).

We denote the original aos-uf-cma experiment played by \( A \) as \( \text{Exp} \). Consider an arbitrary adversary \( A \) that is run in \( \text{Exp} \). At the key-generation stage the challenger picks a random secret key \( \text{key}(\varepsilon) \), computes the values of \( \text{key}(\cdot) \) on all the nodes in the graph \( T \), and gives a public key \( \text{key}(\bar{\varepsilon}) \) to \( A \). The adversary outputs a forgery \( (M^*[1..t], \text{Sig}^*) \) and with probability \( \text{Adv}_{\text{AOS3}, A}^{\text{aos-uf-cma}}(k) \), \( A \)'s forgery is valid. Let \( u^* = \langle M_1^*, \ldots, M_t^* \rangle \) and parse \( \text{Sig}^* \) as the set \( \{\text{key}^*(x) \mid x \in \{u^*\} \cup \text{Comp}(u^*)\} \). Let \( \text{Desc}(u^*) \) be the set \( \{u^*\} \cup \text{Comp}(u^*) \) including all their (lower) descendants in the graph \( T \). Then the signature on \( u^* \) defines \( \text{key}^*(\cdot) \) on the set \( \text{Desc}(u^*) \). We define the event

\[\text{FORGE} : \text{The forgery of } A \text{ in experiment } \text{Exp} \text{ is valid,}\]

i.e. \( \text{key}^*(\bar{\varepsilon}) = \text{key}(\bar{\varepsilon}) \). We define \( \text{KEYS} \) as the event that \( \text{key}^*(x) = \text{key}(x) \) on all \( x \in \text{Desc}(u^*) \) in experiment \( \text{Exp} \). Then we have

\[
\text{Adv}_{\text{AOS3}, A}^{\text{aos-uf-cma}}(k) = \Pr[\text{FORGE}] = \Pr[\text{FORGE} \land \text{KEYS}] + \Pr[\text{FORGE} \land \neg \text{KEYS}] = \underbrace{\Pr[\text{FORGE} \land \text{KEYS}]}_{p_1} + \underbrace{\Pr[\text{FORGE} \land \neg \text{KEYS}]}_{p_2}.
\]
Intuitively, we are going to show that in the first case there is an efficient attack on the pseudorandomness of $G$ or the one-wayness of $G_0$ or $G_1$ and in the second case one can break collision-resistance of $H, G_0$ or $G_1$.

We will bound $p_1$ and $p_2$ in turn. First we bound $p_2$. Note that $\neg E$ means that there exists some $x \in \text{Desc}(u^*)$ such that $\text{key}^*(x) \neq \text{key}(x)$. Hence $\text{FORGE} \land \neg \text{KEYS}$ means that among the possible descendants $x \in \text{Desc}(u^*)$ satisfying $\text{key}^*(x) \neq \text{key}(x)$, there must exist some $x$ together with one of its children $y$, such that $\text{key}^*(x) \neq \text{key}(x)$ but $\text{key}^*(y) = \text{key}(y)$ (since we require $\text{key}^*(\tilde{\epsilon}) = \text{key}(\tilde{\epsilon})$).

Consider the following adversary $B$ (attacking the collision resistance of $G_0$, $G_1$ or $H$) that runs the aos-uf-cma experiment for $A$. $B$ picks a random secret key $\text{key}(\epsilon)$, recursively computes $\text{key}(x)$ for all nodes $x$ in the graph $T$, and returns the public key $\text{key}(\tilde{\epsilon})$ to $A$. The input of $A$ is distributed identically to the experiment $\text{Exp}_{\text{aos-uf-cma}}(A, k)$. Thus with probability $p_2$, $A$ outputs a valid forgery $\{\text{key}^*(x) \mid x \in \{u^*\} \cup \text{Comp}(u^*)\}$ for some node $u^* \in UT$ such that there exists a node $x \in T$ (x is some descendant $x$ of $\{u^*\} \cup \text{Comp}(u^*)$) and its child $y$ such that $\text{key}^*(x) \neq \text{key}(x)$ but $\text{key}^*(y) = \text{key}(y)$. Such an $x$ can be identified by $B$ in the same time as verifying a signature. If $x \in UT$ then $\{\text{key}^*(x), \text{key}(x)\}$ is a collision either for $G_0$ or $G_1$ ($G_i(\text{key}^*(x)) = G_i(\text{key}(x))$ for some $i \in \{0, 1\}$). Otherwise let $x'$ be another parent of $y$, then

$$H(\text{key}^*(x), \text{key}^*(x')) = H(\text{key}(x), \text{key}(x'))$$

and $\{(\text{key}^*(x), \text{key}^*(x')), (\text{key}(x), \text{key}(x'))\}$ is a collision for $H(\cdot, \cdot)$. Therefore

$$p_2 \leq \text{Adv}^c_{H,B}(k) + \text{Adv}^c_{G_0,B}(k) + \text{Adv}^c_{G_1,B}(k).$$

We now proceed to bounding $p_1$. First consider a single adversary $C_0$ participating in the oneway experiments for both $G_0$ and $G_1$ simultaneously. The adversary $C_0$ will run $A$ and in the case $A$ successfully forges $\text{AOS3}$, she will win in either one of the experiments (that is, finding a preimage for $G_0$ or $G_1$ of a given input value $C$). Second, consider an adversary $C_1$ participating in the same oneway experiment and that only slightly differs from the description of adversary $C_0$. The adversary $C_1$ will be used to bound probability $p_1$. 

Finally, using a hybrid argument, we will connect the two adversaries by showing that the views of adversary $A$ when run by $C_0$ and $C_1$ in the execution of the experiment are computational indistinguishable assuming $G$ is a pseudorandom generator.

We describe the two adversaries $C_0$ and $C_1$ as follows:

**Adversary $C_i(1^k, c)$:**

- $\text{key}(\varepsilon) \overset{\$}{\leftarrow} \{0, 1\}^k$ and define $\text{key}(\cdot)$ recursively on all the descendants.

- Pick a random node $u^+ = \langle M^+_1, \ldots, M^+_t \rangle \overset{\$}{\leftarrow} UT \setminus \{\varepsilon\} \cup \{d\text{-th level of } LT\}$. If $i = 0$ then redefine $\text{key}(u^+) \leftarrow c$ else do nothing.

- Run $A(1^k, \text{key}(\varepsilon))$ and answer all the $\text{SIGN}(M[1..t])$ queries made by $A$ as follows:
  - If $(M[1..t] = M^+[1..t])$ and $(t < n)$ then return abort.
  - Else set $u = \langle M_1, \ldots, M_t \rangle$; return $\text{Sig} = \{\text{key}(x) \mid x \in \{u\} \cup \text{Comp}(u)\}$.

- Eventually $A$ terminates and outputs a forgery $\text{Sig}^*$ for some message $M^*[1..t]$.

- Denote $\langle M^*_1, \ldots, M^*_t \rangle$ as $u^*$; parse $\text{Sig}^*$ as $\{\text{key}^*(x) \mid x \in u^* \cup \text{Comp}(u^*)\}$.

- If $M^*[1..t] = M^+[1..n - 1]$ then return $\text{key}^*(u^*)$ else return abort.

For $i \in \{0, 1\}$ denote by $\text{Exp}_i$ the simultaneous one-way experiment when run under adversary $C_i$. We define the two events

- $\text{ABORT}_{C_i}$: Adversary $C_i$ aborts in experiment $\text{Exp}_i$.

- $\text{KEYS}_{C_i}$: $\text{key}^*(x) = \text{key}(x)$ for all $x \in \text{Desc}(u^*)$ in experiment $\text{Exp}_i$.

Furthermore, we define the two probabilities

- $q_i = \Pr[\text{KEYS}_{C_i} \land \neg\text{ABORT}_{C_i}]$

in experiment $\text{Exp}_i$. Intuitively, in the remainder of the proof we will show that the value $q_0$ is related to the one-wayness of $G_0$ and $G_1$, that $q_1$ can be lower bounded using $p_1$, and that $|q_0 - q_1|$ is small.

Consider the experiment $\text{Exp}_0$. Assuming $C_0$ does not abort, $u^+$ is a child of $u^*$ and hence we have $G_{M^+_n}(\text{key}^*(u^*)) = \text{key}(u^+) = c$ if the event $\text{KEYS}_{C_0}$ happens. Hence
adversary $C_0$ outputs $key^*(u^*)$ which is valid preimage of $G_{M^+}(c)$:

$$q_0 \leq \text{Adv}_G^{1\text{-way}}(k) + \text{Adv}_G^{1\text{-way}}(k)$$

in experiment $\text{Exp}_0$. We will now lower bound probability $q_1$ in experiment $\text{Exp}_1$.

**Claim 4.2.6** $q_1 \geq p_1/(3 \cdot 2^d)$.

We rewrite $q_1$ as

$$q_1 = \Pr [\text{KEYS}_{C_1} | \neg\text{ABORT}_{C_1}] \cdot \Pr [\neg\text{ABORT}_{C_1}].$$

The event $\neg\text{ABORT}_{C_1}$ in $\text{Exp}_1$ happens if and only if the right value for $u^+$ was guessed by $C_1$, i.e. if $u^+$ is a child of $u^*$. Hence, assuming $C_1$ does not abort, the view of $A$ in experiment $\text{Exp}_1$ is identically distributed to the view of $A$ the experiment $\text{Exp}$. Hence with probability at least $p_1$, $A$ outputs a valid forgery $(u^*, \text{Sig}^*)$ such that $key^*(x) = key(x)$ on all descendants $x \in \text{Desc}(u^*)$ in $\text{Exp}_1$:

$$\Pr [\text{KEYS}_{C_1} | \neg\text{ABORT}_{C_1}] \geq \Pr [\text{FORGE} \wedge \text{KEYS}] = p_1$$

Since the choice of $u^+$ in the experiment $\text{Exp}_1$ is totally independent from the inputs of $A$ (and $u^+$ is chosen randomly from the set of $3 \cdot 2^d - 2$ nodes), we have

$$\Pr [\neg\text{ABORT}_{C_1}] = 1/(3 \cdot 2^d - 2) \geq 1/(3 \cdot 2^d).$$

This completes the proof of the claim.

For $i \in \{0, 1\}$ define $\text{View}_i$ to be the view of adversary $A$ in experiment $\text{Exp}_i$. The following lemma establishes that the views of $A$ in the experiments $\text{Exp}_0$ and $\text{Exp}_1$ are computationally indistinguishable.

**Lemma 4.2.7** $A$ cannot distinguish between the two distributions $\text{View}_0$ and $\text{View}_1$ better than with probability $2d \cdot \text{Adv}^{\text{prg}}_{G,D}(k)$ for some adversary $D$ running in about the same time as $A$. 
91

We will prove the lemma later. As an implication of Lemma 4.2.7 we get that \(|q_0 - q_1| \leq 2d \cdot \text{Adv}^{prg}_{G,D}(k)\). Thus

\[
p_1 \leq 3 \cdot 2^d \cdot q_1 \\
\leq 3 \cdot 2^d (q_0 + 2d \cdot \text{Adv}^{prg}_{G,D}(k)) \\
\leq 3 \cdot 2^d (\text{Adv}_{1-way}^{G_0,C}(k) + \text{Adv}_{1-way}^{G_1,C}(k)) + 2d \cdot \text{Adv}^{prg}_{G,D}(k)\
\]

Combining the bounds on \(p_1\) and \(p_2\) we complete the proof of the theorem:

\[
p_1 + p_2 \leq \sum_{S \in \{H,G_0,G_1\}} \text{Adv}^{cr}_{S,B}(k) + \\
+ 3 \cdot 2^d \left( \sum_{S \in \{G_0,G_1\}} \text{Adv}_{1-way}^{S,C}(k) + 2d \cdot \text{Adv}^{prg}_{G,D}(k) \right).
\]

**Proof of Lemma 4.2.7:** Each of the experiments \(\text{Exp}_0\) and \(\text{Exp}_1\) is completely determined by the values of \(\text{key}(\cdot)\) on the following subset of nodes \(R \subseteq T\):

\[
R = \{\langle M_1^+, \ldots, M_j^+ \rangle \mid j = 1, \ldots, n\} \cup \{\langle M_1^+, \ldots, M_{j-1}^+, \neg M_j^+ \rangle \mid j = 1, \ldots, n\}
\]

In \(\text{Exp}_1\), \(\text{key}(\varepsilon)\) is chosen at random and the keys on the rest of \(R\) is generated by iterating \(G(\cdot)\). On the other hand, in \(\text{Exp}_0\) the values of \(\text{key}(\cdot)\) on \(R\) are defined in the same way except for \(\text{key}((M_1^+, \ldots, M_n^+))\), which is replaced by the challenge string for the onewayness experiment. Our goal is to show that from adversary \(A\)'s view, these two experiments are computationally indistinguishable.

We define a sequence of \(n\) hybrid adversaries \(C_1[1], \ldots, C_1[n]\) as follows: Let \(C_1[1] = C_1\) and the \(i\)-th adversary \(C_1[i]\) behaves like \(C_1\) except that it assigns random keys \(\text{key}(\cdot)\) on the nodes \(\{\langle M_1^+, \ldots, M_j^+ \rangle, \langle M_1^+, \ldots, M_{j-1}^+, \neg M_j^+ \rangle \mid j = 1, \ldots, i\}\) and uses \(G(\cdot)\) to compute \(\text{key}(\cdot)\) on the rest of the nodes. The only difference between adversaries \(C_1[i]\) and \(C_1[i + 1]\) is that in the first case \(\text{key}((M_1^+, \ldots, M_i^+)), \text{key}((M_1^+, \ldots, \neg M_i^+))\) are pseudorandom and in the second case they are perfectly random. Let \(\text{Exp}_1[i]\) be the experiment when run under adversary \(C_1[i]\) and let \(\text{View}_1[i]\) be the random variable describing the view of adversary \(A\) when run in experiment \(\text{Exp}_1[i]\). Then, for each
1 \leq i \leq n - 1, \mathcal{A} cannot distinguish between View_1[i] and View_1[i + 1] any better than with probability \text{Adv}^{\text{prog}}_{G,D}(k) for some adversary D who runs in the time needed to run \mathcal{A}.

Next, consider a new adversary C'_0 which is defined as C_0, just key(\langle M_1^+, \ldots, M_n^+ \rangle) is replaced by a random string from \{0, 1\}^k. Analogously define Exp'_0 and View'_0. Since G(\cdot) is a pseudorandom generator, \mathcal{A} cannot distinguish between the View_0 and View'_0 any better than with probability \text{Adv}^{\text{prog}}_{G,D}(k).

Analogous to the first sequence of hybrid adversaries, consider a sequence of adversaries C'_0[1], \ldots, C'_0[n]. Here C'_0[1] = C'_0 and the i-th adversary C'_0[i] is identical to C_1[i], except the value key(\langle M_1^+, \ldots, M_n^+ \rangle) is replaced by a truly random string. Again define Exp'_0[i] and View'_0[i] as above. A similar argument shows that \mathcal{A} cannot distinguish between View'_0[i] and View'_0[i + 1] any better than with probability \text{Adv}^{\text{prog}}_{G,D}(k). Finally, the experiments Exp'_0[n] and Exp_1[n] are identical so are View'_0[n] and View_1[n]. This establishes a sequence of hybrids from View_0 to View_1:

View_0 \approx View'_0 \equiv View'_0[1] \approx \ldots \approx View'_0[n] \equiv View_1[n] \approx \ldots \approx View_1[1] \equiv View_1,

where \approx means that the distributions of the two random variables are computational indistinguishable and \equiv means they are identical. \mathcal{A} cannot distinguish between any consecutive pair of the views any better than with probability \text{Adv}^{\text{prog}}_{G,D}(k), therefore total probability that \mathcal{A} will distinguish between View_0 and View_1 is no more than

\[ 2n \cdot \text{Adv}^{\text{prog}}_{G,D}(k) \leq 2d \cdot \text{Adv}^{\text{prog}}_{G,D}(k). \]

\[ \blacksquare \]

### 4.2.7 AOS via One-time Signatures

We observe that we can combine the ideas of certificate-based AOS (Section 4.2.1) and hash-tree AOS (Section 4.2.5) to gain a more efficient append-only signature scheme when the message space is small. Assume the message space AOS.MSpace consists
of $m$ elements. Then we can use our certificate-based construction $AOS_1$ instantiated with a $m$-time signature scheme. $m$-time signatures can be efficiently constructed using hash-trees (see [20], [37], [47], and [56] for the definition and efficient constructions of $m$-time signatures). In addition, the security proof of $AOS_1$ guarantees unforgeability if $SGN$ is at least an $|AOS.MSpace|$-time signature scheme. Note that in contrast to $AOS_3$, the length of the AOS messages in this construction is unbounded.

### 4.3 Relations between HIBS and AOS

In this section, we show that the concepts of AOS and Hierarchical Identity-based Signatures (HIBS) are in fact equivalent. Before we present the poly-time reductions between the AOS and HIBS, we first review related concepts and relationships.

In Identity-based Signature (IBS) schemes, the identity of a sender (for example, an email address) is used as a public key for verification of the signature. This approach assumes the existence of a trusted party (the certificate authority), which assigns secret keys to all users. The certificate authority has a pair of keys: a master public key and a master secret key; the master secret key is used to delegate keys to the users and the master public key is used for signature verification. Anyone can verify signatures on messages signed by any user, knowing only the master public key and the identity of that user.

Hierarchical Identity-based Signature (HIBS) schemes are a natural generalization of IBS to the setting with a hierarchical organization of users. Assume that the users are structured in a tree with a root being the certificate authority. Descendants of a user are the usernames that contain this user’s name as a prefix; the canonical example involves domain names. In HIBS, each individual user can play the role of a certificate authority and can delegate secret keys to his descendants. As in IBS, a secret key allows a user to sign arbitrary messages such that anyone is able to verify the signature knowing only the

---

2 Similar ideas were used by Abdalla and Reyzin [1] who suggested how to improve the efficiency of binary certification method for constructing forward-secure signatures (see also Bellare and Miner [11]).
Hierarchical Identity-based Encryption (HIBE) assumes the same hierarchy of users as HIBS and provides encryption/decryption mechanisms rather than signatures. In HIBE, anyone can encrypt data to any user in the hierarchy knowing only the user’s identity and the master public key; the ciphertext can only be decrypted using the user’s secret key.

As noted by Naor (see Section 6 of [23]), any IBE scheme can readily be converted into a public key signature scheme (by interpreting user identities of IBE as messages for a regular signature scheme and defining the signature of a message to be the secret key associated with the corresponding identity). Similarly, any HIBE scheme can be transformed into a HIBS scheme. This was sketched by Gentry and Silverberg [29], giving the construction for a HIBS scheme. We note that the converse (transforming HIBS into HIBE) is not known to be possible. Related to these are Forward-secure Signature (FSS) schemes, which modify a secret key over time (while the public key remains the same) such that exposure of the secret key at a certain time period does not allow forgery of signatures from previous time periods. (See [11] for an exact definition of FSS.) In [25] it is proved that HIBE implies Forward-secure Encryption (FSE) (which is defined as FSS with signing replaced by encryption). Using the same construction it is easy to show that HIBS implies FSS (as explicitly noted in [26]). More precisely, a HIBS scheme of depth $d$ can be used to construct a forward-secure scheme providing security for $2^d$ time steps (using a tree-based construction). The converse (transforming FSS into HIBS) is not known to be possible. Thus, relating HIBE, HIBS, AOS, and FSS, we obtain the following hierarchy:

$$
\text{HIBE} \Rightarrow \text{HIBS} \Leftrightarrow \text{AOS} \Rightarrow \text{FSS}
$$

In particular, given a secure HIBE scheme, we can construct a secure AOS scheme and given a secure AOS scheme, it is easy to obtain a secure FSS scheme.
4.3.1 Definition of HIBS

We recall the syntax of Hierarchical Identity-based Signature (HIBS) schemes and the appropriate notions of unforgeability. Let HIBS.IDSpace be any set of identities (typically $\{0, 1\}^*$). For an integer $n \geq 0$, a username at the level $n$ in the tree (called hierarchical identity of depth $n$) is an (ordered) $n$-tuple of identities written as $I[1..n] = (I_1, I_2, \ldots, I_n)$ with each $I_i \in$ HIBS.IDSpace. The special case of $n = 0$ is the root identity, denoted as $I[1..0]$ or $\varepsilon$. Further on, we will refer to strings from HIBS.IDSpace as identities and to $n$-tuples of them as hierarchical identities. We use the symbol $\sqsubseteq$ to denote the prefix relation over the set of hierarchical identities: for a given hierarchical identity $I[1..n] = (I_1, I_2, \ldots, I_n)$, any hierarchical identity from the set $\{I[1..i], 0 \leq i \leq n\}$ is its prefix. Note that the root identity $\varepsilon$ is a prefix of any other hierarchical identity.

A HIBS scheme $\mathcal{HIBS}$ with respect to the message space HIBS.MSpace and the identity space HIBS.IDSpace is made up of four algorithms: a setup algorithm HIBS.Setup, a key delegation algorithm HIBS.KeyDel, a signature algorithm HIBS.Sign, and a verification algorithm HIBS.Vfy.

- HIBS.Setup (the setup algorithm) takes as input a security parameter and generates the master public key HIBS.pk of the scheme and the secret key of the root identity HIBS.SK[\varepsilon] (the master secret key).

- HIBS.KeyDel (the key delegation algorithm) takes as input a hierarchical identity $I[1..n] = (I_1, \ldots, I_n)$, its associated secret key HIBS.SK[$I[1..n]$], and an identity $I_{n+1} \in$ HIBS.IDSpace of its child. It returns a secret key HIBS.SK[$I[1..n + 1]$] associated with the new hierarchical identity $I[1..n + 1] = (I_1, \ldots, I_n, I_{n+1})$.

- HIBS.Sign (the signing algorithm) takes a hierarchical identity $I[1..n]$, the associated secret key HIBS.SK[$I[1..n]$], and a message $M \in$ HIBS.MSpace. It computes a signature on this message $M$ with respect to this identity.

- HIBS.Vfy (the verification algorithm) takes the master public key HIBS.pk, a hi-
Hierarchical identity \( I[1..n] \), a message \( M \), and a signature \( \text{sig} \). It outputs \text{true} or \text{false} depending on whether \( \text{sig} \) is a valid signature of \( M \) signed by hierarchical identity \( I[1..n] \).

All these algorithms can be randomized. All of them must be polynomial-time in the security parameter. Moreover, it is required that for all pairs \((\text{HIBS.pk}, \text{HIBS.SK}[\varepsilon])\) of master public and secret keys output by \text{HIBS.Setup}, and for all messages \( M \in \text{HIBS.MSpace} \), hierarchical identities \( I[1..n] \) and associated secret keys \( \text{HIBS.SK}[I[1..n]] \) (recursively generated from the secret key \( \text{HIBS.SK}[\varepsilon] \) using the \text{HIBS.KeyDel} algorithm),

\[
\text{HIBS.Vfy}(\text{HIBS.pk}, I[1..n], \text{HIBS.Sign}(\text{HIBS.SK}[I[1..n]], I[1..n]), M, M) = \text{true}.
\]

Unforgeability of the HIBS scheme \( \text{HIBS} \) under chosen-plaintext attacks is formally defined as follows:

**Definition 4.3.1 [HIBS-UF-CMA]** Let \( \text{HIBS} = (\text{HIBS.Setup}, \text{HIBS.KeyDel}, \text{HIBS.Sign}, \text{HIBS.Vfy}) \) be a hierarchical identity-based signature scheme, let \( k \) be the security parameter, and let \( A \) be an adversary. We consider the experiment:

**Experiment** \( \text{Exp}_{\text{HIBS}\cdot A}^{\text{hibs-uf-cma}}(k) \)

\[
\begin{aligned}
\text{IDSet} &\leftarrow \emptyset; (\text{HIBS.pk}, \text{HIBS.SK}[\varepsilon]) \leftarrow \text{HIBS.Setup}(1^k) \\
(I[1..n], M, \text{sig}) &\leftarrow A^{\text{CORRUPT}(), \text{SIGN}(), \cdot}(\text{HIBS.pk}) \\
\text{if } &\text{HIBS.Vfy}(I[1..n], M, \text{sig}) = \text{true} \\
\text{and } &\forall J[1..j] \subseteq I[1..n]: J[1..j] \notin \text{IDSet} \\
\text{and } &(I[1..n], M) \notin \text{MSGSet} \\
\text{then return } 1 \text{ else return } 0
\end{aligned}
\]

**Oracle** \( \text{CORRUPT}(I[1..n]) \)

\[
\text{IDSet} \leftarrow \text{IDSet} \cup \{I[1..n]\}
\]

**return** \( \text{EXTRACT}(I[1..n]) \)

**Oracle** \( \text{SIGN}(I[1..n], M) \)
\[ \text{MSGSet} \leftarrow \text{MSGSet} \cup \{(I[1..n], M)\} \]

\[ \text{HIBS(SK}[I[1..n]] \leftarrow \text{EXTRACT}(I[1..n]) \]

\[ \text{return HIBS.Sign(HIBS.SK}[I[1..n]], I[1..n], M) \]

**Oracle** \text{EXTRACT}(I[1..i]) // defined recursively

\[ \text{if } i = 0 \text{ return HIBS.SK}[\varepsilon] \]

\[ \text{else if HIBS.SK}[I[1..i]] = \text{defined} \]

\[ \text{then return HIBS.SK}[I[1..i]] \]

\[ \text{else HIBS.SK}[I[1..i]] \leftarrow \ldots \]

\[ \ldots \text{HIBS.KeyDel(HIBS.pk, I}[1..i-1], \text{EXTRACT}(I[1..i-1]), I_i) \]

\[ \text{return HIBS.SK}[I[1..i]] \]

The hibs-uf-cma-advantage of an adversary \( \mathcal{A} \) in breaking the security of the scheme \( \mathcal{HIBS} \) is defined as

\[ \text{Adv}^{\text{hibs-uf-cma}}_{\mathcal{HIBS}, \mathcal{A}}(k) = \text{Pr} \left[ \text{Exp}^{\text{hibs-uf-cma}}_{\mathcal{HIBS}, \mathcal{A}}(k) = 1 \right], \]

and \( \mathcal{HIBS} \) is said to be existentially unforgeable under chosen message attacks (and shortly, hibs-uf-cma-secure) if the above advantage is a negligible function in \( k \) for all polynomial-time adversaries \( \mathcal{A} \). Note that the adversary is given access to the two oracles \( \text{CORRUPT}(\cdot) \) and \( \text{SIGN}(\cdot, \cdot) \), not to the oracle \( \text{EXTRACT}(\cdot) \). The latter one is only used internally by the experiment.

### 4.3.2 Constructing AOS from HIBS

The idea of the reduction is as follows. We set \( \text{AOS.MSpace} = \mathcal{HIBS.IDSpace} \) and associate an AOS message \((M_1, \ldots, M_n)\) of length \( n \) with the hierarchical identity \( I[1..n] = (M_1, \ldots, M_n) \) of depth \( n \). We then define the signature of this message as the secret key \( \text{HIBS.SK}[I[1..n]] \) of the hierarchical identity \( I[1..n] \).

Given the above analogy between signatures of messages and secret keys of hierarchical identities, we construct an AOS scheme given a HIBS scheme as follows.
Appending to a given signature in $AOS$ is done using key delegation in $HIBS$. The verification of an AOS signature $HIBS.SK[I[1..n]]$ is done by signing a random message $M \in HIBS.MSpace$ under the secret key $HIBS.SK[I[1..n]]$ and verifying that the resulting signature is valid.

**Construction 4.3.2** Given a HIBS scheme $HIBS = (HIBS.Setup, HIBS.KeyDel, HIBS.Sign, HIBS.Vfy)$, we construct an AOS scheme $AOS = (AOS.Setup, AOS.Append, AOS.Vfy)$ as follows:

- **AOS.Setup**: Run the $HIBS.Setup$ algorithm to generate a pair of keys $(HIBS.pk, HIBS.SK[ε])$ and output $(HIBS.pk, HIBS.SK[ε])$ as the key pair for $AOS$. Here $HIBS.SK[ε]$ is the signature of an empty message $ε$.

- **AOS.Append**: Given the public key $AOS.pk$, signature $Sig[M[1..n]]$ of the message $M[1..n]$, and the message $M_n$ to append, the signature on $Sig[M[1..n+1]]$ is returned as $Sig[M[1..n+1]] ← HIBS.KeyDel(HIBS.pk, Sig[M[1..n]], M_{n+1})$.

- **AOS.Vfy**: Given a public key $AOS.pk$, a message $M[1..n]$, and the signature $Sig[I[1..n]]$, the verification algorithm first signs a random message $M \in HIBS.MSpace$ under hierarchical identity $M[1..n]$ using $Sig[I[1..n]]$ as a secret key of $M[1..n]$ in $HIBS$:

$$\text{sig} \leftarrow HIBS.Sign(I[1..n], Sig[I[1..n]], M).$$

Then it outputs the result of the $HIBS$ verification $HIBS.Vfy(HIBS.pk, I[1..n], M, \text{sig})$.

**Theorem 4.3.3** If the HIBS scheme $HIBS = (HIBS.Setup, HIBS.KeyDel, HIBS.Sign, HIBS.Vfy)$ is hibs-uf-cma secure, then the AOS scheme from Construction 4.3.2 is aos-uf-cma secure.

We omit the proof of Theorem 4.3.3.
4.3.3 Constructing HIBS from AOS

A naive approach to building a HIBS scheme $\mathcal{HIBS}$ from an AOS scheme $\mathcal{AOS}$ is to first append all the identities and then to append a message to be signed. That is, both the identity space $\mathcal{HIBS}.\text{IDSpace}$ and the message space $\mathcal{HIBS}.\text{MSpace}$ of $\mathcal{HIBS}$ are subsets of the message space $\mathcal{AOS}.\text{MSpace}$ of the $\mathcal{AOS}$ scheme. The secret key of a hierarchical identity $I[1..n]$ in this HIBS scheme is exactly the AOS signature of $I[1..n]$ viewed as the AOS message. Delegation in HIBS is equivalent to appending in AOS. Signing a message $M$ with respect to a hierarchical identity $I[1..n]$ is defined by appending $M$ to the AOS signature of $I[1..n]$.

This naive construction is secure only if $\mathcal{HIBS}.\text{IDSpace}$ and $\mathcal{HIBS}.\text{MSpace}$ are disjoint subsets of $\mathcal{AOS}.\text{MSpace}$. If there is some identity $J$ which itself is a valid message, the security of the HIBS scheme can be broken even if its corresponding AOS is secure. An adversary could query the HIBS signing oracle with a message $J$ and some hierarchical identity $I[1..n]$. The resulting signature equals the secret key for the hierarchical identity $(I_1, \ldots, I_n, J)$, which violates the security of the HIBS scheme.

Our idea to overcome this problem is to insert a unique identifier between identities and messages. Let $\mathcal{AOS} = (\mathcal{AOS}.\text{Setup}, \mathcal{AOS}.\text{Append}, \mathcal{AOS}.\text{Vfy})$ be a secure AOS scheme with message space $\mathcal{AOS}.\text{MSpace}$. Let $\mathcal{HIBS}.\text{IDSpace}$ and $\mathcal{HIBS}.\text{MSpace}$ be arbitrary subsets of $\mathcal{AOS}.\text{MSpace}$ such that there is some symbol $\Delta$ from the AOS message space which is not a valid identity for the HIBS scheme ($\Delta$ can still be in the HIBS message space). Then we can construct a secure HIBS scheme $\mathcal{HIBS} = (\mathcal{HIBS}.\text{Setup}, \mathcal{HIBS}.\text{KeyDel}, \mathcal{HIBS}.\text{Sign}, \mathcal{HIBS}.\text{Vfy})$ with identity space $\mathcal{HIBS}.\text{IDSpace}$ and message space $\mathcal{HIBS}.\text{MSpace}$ as follows:

Construction 4.3.4 $\mathcal{HIBS} = (\mathcal{HIBS}.\text{Setup}, \mathcal{HIBS}.\text{KeyDel}, \mathcal{HIBS}.\text{Sign}, \mathcal{HIBS}.\text{Vfy})$:

- $\mathcal{HIBS}.\text{Setup}(1^k)$: Run the $\mathcal{AOS}.\text{Setup}(1^k)$ algorithm to generate a pair $(\mathcal{AOS}.\text{pk}, \text{Sig}[\varepsilon])$ and output it as the master public/private key pair for $\mathcal{HIBS}$.

- $\mathcal{HIBS}.\text{KeyDel}(\mathcal{HIBS}.\text{pk}, \mathcal{HIBS}.\text{SK}[I[1..n]], I_{n+1})$: Given the master public key, the secret key of hierarchical identity $I[1..n]$, and a new identity $I_{n+1}$, the delegation
algorithm interprets \( \text{HIBS.SK}[I[1..n]] \) as an \( \text{AOS} \) signature of \( I[1..n] \). It appends to the signature a message \( I_{n+1} \) and outputs the resulting signature as the secret key of \( I[1..n+1] \):

\[
\text{HIBS.SK}[I[1..n+1]] \leftarrow \text{AOS.Append}(\text{HIBS.pk}, \text{HIBS.SK}[I[1..n]], I_n).
\]

- \( \text{HIBS.Sign}(\text{HIBS.pk}, \text{HIBS.SK}[I_n], M) \): Given a master public key, a secret key of the hierarchical identity \( I[1..n] \), and a message \( M \), the sign algorithm for \( \text{HIBS} \) interprets \( \text{HIBS.SK}[I[1..n]] \) as an \( \text{AOS} \) signature of \( I[1..n] \). It appends a symbol \( \Delta \) to \( \text{HIBS.SK}[I[1..n]] \) and then appends a message \( M \) to the resulting AOS signature to get the final signature \( \text{sig} \):

\[
\text{sig} \leftarrow \text{AOS.Append}(\text{HIBS.pk, AOS.Append}(\text{HIBS.pk, HIBS.SK}[I[1..n]], \Delta), M).
\]

- \( \text{HIBS.Vfy}(\text{HIBS.pk, I[1..n], M, sig}) \): Given a master public key, a hierarchical identity \( I[1..n] \), a signature \( \text{sig} \), and a message \( M \), the verification algorithm for \( \text{HIBS} \) returns the output of

\[
\text{AOS.Vfy}(\text{HIBS.pk, (I_1, \ldots, I_n, \Delta, M, \text{sig})}.
\]

**Theorem 4.3.5** If the AOS scheme \( \text{AOS} = (\text{AOS.Setup}, \text{AOS.Append, AOS.Vfy}) \) is \( \text{aos-uf-cma secure} \), then HIBS scheme \( \text{HIBS} \) from Construction 4.3.4 is \( \text{hibs-uf-cma secure} \).

**Proof:** Given an adversary \( A \) attacking the hibs-uf-cma security of the \( \text{HIBS} \) scheme with success probability \( \text{Adv}^{\text{hibs-uf-cma}}_{\text{HIBS, A}}(k) \), we can construct an adversary \( B \) that attacks the aos-uf-cma security of \( \text{AOS} \) with success probability

\[
\text{Adv}^{\text{aos-uf-cma}}_{\text{AOS, B}}(k) \geq \text{Adv}^{\text{hibs-uf-cma}}_{\text{HIBS, A}}(k).
\]

As in Definition 4.1.1, adversary \( B \) gets input a public key AOS.pk for the \( \text{AOS} \) scheme. \( B \) runs the HIBS-UF-CMA experiment against \( \text{HIBS} \) as well as an instance of adversary
\( A \). \( B \) gives as input to \( A \) the master public key \( \text{HIBS.pk} = \text{AOS.pk} \). Adversary \( B \) answers the oracle queries of adversary \( A \) using the oracle \( \text{AOSSIGN}(\cdot) \) for \( \text{AOS} \) as follows:

**Corrupt**\((I[1..n])\):

\[
\text{HIBS.SK}[I[1..n]] \leftarrow \text{AOSSIGN}(I[1..n])
\]

return \( \text{HIBS.SK}[I[1..n]] \) to \( A \).

**Sign**\((I[1..n], M)\):

\[
\text{sig0} \leftarrow \text{AOSSIGN}(I[1..n])
\]

\[
\text{sig1} \leftarrow AOS.\text{Append}(\text{AOS.pk}, \text{sig0}, \Delta)
\]

\[
\text{sig2} \leftarrow AOS.\text{Append}(\text{AOS.pk}, \text{sig1}, M)
\]

return \( \text{sig2} \) to \( A \).

Eventually, adversary \( A \) halts and outputs the triple \((I^*[1..n], M^*, \text{sig}^*)\), which consists of a target hierarchical identity \( I^*[1..n] \), a message \( M^* \), and a forged signature \( \text{sig}^* \). Adversary \( B \) then outputs \( \text{sig}^* \) as the forgery of message \( M^*[1..n+2] = (I^*_1, \ldots, I^*_n, \Delta, M^*) \). This completes the description of the simulation.

It is easy to see that \( B \) perfectly simulates the two oracles **Corrupt**\((\cdot)\) and **Sign**\((\cdot, \cdot)\); that is, \( B \)'s responses on \( A \)'s queries are distributed exactly as in the true HIBS-UF-CMA experiment.

Note that if \( \text{sig}^* \) is a valid signature of \( M^* \) with respect to the hierarchical identity \( I^*[1..n] \), then \( \text{sig}^* \) is also a valid AOS signature on the message \( M^*[1..n+2] \).

The fact that \( \Delta \notin \text{HIBS.IDSpace} \) ensures that the message \( M^*[1..n+2] \) was never queried by \( B \) to the oracle \( \text{AOSSIGN}(\cdot) \) when simulating the oracle **Corrupt**\((\cdot)\). Furthermore, if \( A \)'s forgery is valid, no prefix of the hierarchical identity \( I^*[1..n] \) can be queried to the oracle **Corrupt**\((\cdot)\) by \( A \) and hence no prefix of \( M^*[1..n+2] \) was queried to the oracle \( \text{AOSSIGN}(\cdot) \) by \( B \). Also, \( A \) is not allowed to call oracle **Sign**\((\cdot)\) for the
tuple \((I^*[1..n], M^*)\). This ensures that no prefix of \(M^*[1..n+2]\) was ever queried to oracle \(\text{AOSSign}(\cdot)\) when simulating oracle \(\text{SIGN}(\cdot, \cdot)\). Thus, whenever \(A\) outputs a valid forgery, \(B\) wins AOS-UF-CMA game against \(\mathcal{AOS}\).

The above reduction uses the existence of a symbol \(\Delta\) s.t. \(\Delta \in \text{AOS.MSpace}\) but \(\Delta \not\in \text{HIBS.IDSpace}\). For the case that \(\text{HIBS.IDSpace} = \text{AOS.MSpace}\), there is an alternative way of constructing a HIBS from an AOS scheme. We sketch the construction for the interesting binary case—that is, for \(\text{HIBS.IDSpace} = \text{AOS.MSpace} = \text{HIBS.MSpace} = \{0, 1\}\).

The HIBS secret key of the hierarchical identity \(I[1..n]\) is then defined to be the AOS signature of \((I_1, 0, I_2, 0, \ldots, I_n, 0)\). The HIBS signature of \(M\) under the hierarchical identity \(I[1..n]\) is defined as the AOS signature of \((I_1, 0, \ldots, I_n, 0, M, 1)\). The security proof of the resulting HIBS scheme is a natural modification of the proof of Theorem 4.3.5.

### 4.3.4 Discussion

Given the equivalence between the AOS and HIBS schemes, one can easily transform all our constructions in Section 4.2 into provably secure HIBS schemes. Note that since the reductions are tight, an efficient AOS implies an efficient HIBS and vice-versa. The advantages of this indirect approach to designing HIBS are twofold. First, AOS is a much simpler primitive than HIBS; security proofs for AOS schemes are easier to carry out than those for HIBS. Second, some of the tricks used in efficient AOS schemes (for example, the hash-tree based construction) could yield more efficient HIBS constructions.

The certificate-based AOS scheme (Section 4.2.1) thus naturally transforms a public-key signature scheme into a HIBS scheme. The certificate-based approach to constructing HIBS schemes was mentioned in [29] although this fact appears not to be widely known [26] and we were not able to find any further studies of certificate-based HIBS in the literature. To the best of our knowledge, we are the first to prove secu-
rity for this scheme. In contrast to identity-based encryption (which is believed to be hard to implement without use of bilinear maps) such HIBS schemes do not utilize pairings, thereby yielding efficient implementations. Moreover, the security proofs are done without using the Random Oracle model.

4.4 Append-only Message Authentication Schemes

Similarly to append-only signatures, we introduce a primitive for the symmetric-key setting, called Append-only Message Authentication (AOMA) scheme. An AOMA scheme allows two parties, who share a secret key, to send authenticated messages to each other, while also enabling any “intermediate” party (who does not have access to the shared key) to append to the message in a verifiable manner. Thus, given any message and its corresponding authentication tag, it should be possible to efficiently append additional information to the message (and produce a tag for the new message). Still, it should be infeasible to perform any other modifications on the authentication tag. AOMA scheme can be viewed as a secret key variant of AOS: a secret key is needed both for initial authentication tag generation and for verification. Still, the append operation is public; any party given an AOMA authentication tag on a message \((M_1, \ldots, M_n)\) can produce a valid AOMA tag on a message \((M_1, \ldots, M_n, M_{n+1})\).

We proceed with a formal definition of an append-only message authentication scheme and then provide a construction of a secure AOMA scheme from an arbitrary block cipher.

4.4.1 Definitions

An append-only message authentication (AOMA) scheme with respect to the message space \(\text{AOMA.MSpace}\) is a collection of three algorithms: a setup algorithm \(\text{AOMA.Setup}\), an append algorithm \(\text{AOMA.Append}\), and a verification algorithm \(\text{AOMA.Vfy}\), defined as follows:

- \(\text{AOMA.Setup}\) (the key generation algorithm) takes the security parameter as input
and outputs the secret key $\text{Tag}[\varepsilon]$, which can also be viewed as the authentication tag on the empty message $\varepsilon$.

- **AOMA.Append** (the append algorithm) takes an authentication tag on a message $M[1..n-1] = (M_1, \ldots M_{n-1})$, of length $n-1$, and a symbol $M_n \in \text{AOMA.MSpace}$ and produces a tag on the message $M[1..n] = (M_1, \ldots, M_n)$.

- **AOMA.Vfy** (the verification algorithm) takes the secret key $\text{Tag}[\varepsilon]$, a message $M[1..n]$, and an authentication tag $\text{tag}$, and returns either true or false.

All algorithms can be randomized and all of them must be polynomial-time in the security parameter. Additionally, the scheme should have the property that for any secret key $\text{Tag}[\varepsilon]$ generated by $\text{AOMA.Setup}(1^k)$ and any message $M[1..n] = (M_1, M_2, \ldots, M_n)$, where $n$ is polynomially bounded in the security parameter, the authentication tag on $M[1..n]$ given by

$$
\text{tag} \leftarrow \text{AOMA.Append}(\text{AOMA.Append}(\cdots, \text{AOMA.Append}(\text{Tag}[\varepsilon], M_1), \cdots, M_{n-1}), M_n)
$$

should be accepted by a verification algorithm. That is, $\text{AOMA.Vfy}(\text{Tag}[\varepsilon], M[1..n], \text{tag})$ must return true.

**Definition 4.4.1** [AOMA-UF-CMA] Let $\mathcal{A} = (\text{AOMA.Setup}, \text{AOMA.Append}, \text{AOMA.Vfy})$ be an AOMA scheme, let $k$ be the security parameter, and let $\mathcal{A}$ be an adversary. We consider the experiment:

**Experiment** $\text{Exp}_{\text{AOMA},\mathcal{A}}^{\text{aoma-uf-cma}}(k)$

1. $\text{MSGSet} \leftarrow \emptyset$; $\text{Tag}[\varepsilon] \leftarrow \text{AOMA.Setup}(1^k)$
2. $(M[1..n], \text{tag}) \leftarrow \mathcal{A}^{\text{AOMASign}(\cdot)}(1^k)$
3. if $\text{AOMA.Vfy}(\text{Tag}[\varepsilon], M[1..n], \text{tag}) = \text{true}$
   - and $\forall J[1..j] \subseteq M[1..n] : J[1..j] \not\in \text{MSGSet}$
   - then return 1 else return 0

**Oracle** $\text{AOMASign}(M[1..n])$
\[ \text{MSGSet} \leftarrow \text{MSGSet} \cup \{M[1..n]\} \]
\[ \text{return } \text{EXTRACT}(M[1..n]) \]

Oracle \text{EXTRACT}(M[1..i]) \text{ // defined recursively}
if \( i = 0 \) then return \Tag{ε}
else if \Tag{M[1..i]} = defined
\then return \Tag{M[1..i]}
else \Tag{M[1..i]} \leftarrow \text{AOMA.Append}((\text{EXTRACT}(M[1..i-1]), M_i))
\text{return } \Tag{M[1..i]} \]

The aoma-uf-cma-advantage of an adversary \( A \) in breaking the security of the scheme \( \text{AOMA} \) is defined as
\[ \text{Adv}_{\text{aoma-uf-cma}}(k) = \Pr[\text{Exp}_{\text{AOMA},A}(k) = 1], \]
and \( \text{AOMA} \) is said to be unforgeable under chosen message attacks (aoma-uf-cma-secure) if the above advantage is a negligible function in \( k \) for any polynomial-time adversary \( A \).

Note that in definition of security, adversary \( A \) is given access to oracle \( \text{AOMASIGN}(\cdot) \) but not to oracle \( \text{EXTRACT}(\cdot) \). The latter is used internally by \( \text{AOMASIGN}(\cdot) \) to create intermediate tags.

### 4.4.2 Construction of AOMA scheme

We provide a very simple and efficient construction of an AOMA scheme based on a hash function. Let \( k \) be a security parameter and let \( H : \{0,1\}^k \times \{0,1\}^k \rightarrow \{0,1\}^k \) be a hash function. The scheme \( \text{AOMA}_1 = (\text{AOMA.Setup}, \text{AOMA.Append}, \text{AOMA.Vfy}) \) works over a message space \( \text{AOMA.MSpace} = \{0,1\}^k \) and its constituent algorithms are specified below:

- \( \text{AOMA.Setup}(1^k) : \) Pick \( \tau \) at random from \( \{0,1\}^k \) and assign \( \Tag{ε} \leftarrow \tau \).
Table 4.5  Efficiency of $\mathcal{AOMA}_1$. Here $n$ represents the length of a given message, and $k$ is the security parameter. $H(\cdot, \cdot)$ denotes an invocation of a hash function.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Efficiency of $\mathcal{AOMA}_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag length</td>
<td>$k$ bits</td>
</tr>
<tr>
<td>Setup time</td>
<td>Generating random $k$-bit string</td>
</tr>
<tr>
<td>Append time</td>
<td>$1 \times H(\cdot, \cdot)$</td>
</tr>
<tr>
<td>Verify time</td>
<td>$n \times H(\cdot, \cdot)$</td>
</tr>
</tbody>
</table>

- $\text{AOMA}.\text{Append}(\text{Tag}[M[1..n]], M_{n+1})$: Assign $\tau \leftarrow \text{Tag}[M[1..n]]$ and return a hash value $H(\tau, M_{n+1})$.

- $\text{AOMA}.\text{Vfy}(\text{Tag}[\varepsilon], M[1..n], \text{tag})$: Assign $\tau_0 \leftarrow \text{Tag}[\varepsilon]$. For $i \leftarrow 1 \ldots n$ compute $\tau_i \leftarrow H(\tau_{i-1}, M_{i-1})$. Return true if $\text{tag} = \tau_n$ and false otherwise.

The efficiency of $\mathcal{AOMA}_1$ is given in Table 4.5. Note that in contrast to AOS, our AOMA scheme construction has constant-size tags. We remark that our construction of AOMA scheme resembles the basic cascade construction for PRF by Bellare et al. [4].

We prove our construction to be a secure AOMA scheme in the random oracle model [13]. That is, we assume that $H$ is a truly random function of both arguments and that parties only access $H$ as a black-box.

**Theorem 4.4.2** Assume that a hash function $H$ in the construction of $\mathcal{AOMA}_1$ is modeled by the random oracle. Then, for any (computationally unbounded) adversary $\mathcal{A}$ against $\mathcal{AOMA}_1$

$$\text{Adv}^\text{aoma-uf-cma}_{\mathcal{AOMA}, \mathcal{A}}(k) \leq \frac{N(N + 1) + 2}{2^{k+1}},$$

where $N$ is a total upper bound on the number of $H(\cdot, \cdot)$ queries made by $\mathcal{A}$ plus a total length (in message symbols) of all signing queries made by $\mathcal{A}$ plus a length of the forged message.

**Proof:** Consider any adversary $\mathcal{A}$; without loss of generality we can assume that $\mathcal{A}$ is deterministic. We are going to bound the advantage of $\mathcal{A}$ for any choice of its random coins; the probability in the security experiment being taken only over a random choice of $H$. 
The adversary $\mathcal{A}$ has two ways of accessing the random oracle $H$: it can either directly make a query to $H(\cdot, \cdot)$ or it can obtain values of $H$ via queries to the AOMASign$(\cdot)$ oracle. In the second case, the random oracle is queried by the security experiment and a value of $H$ might be included in the authentication tag returned by the signing oracle. We say that a random oracle is defined on a tuple $(K, M)$ if $(K, M)$ was queried to $H(\cdot, \cdot)$ either by $\mathcal{A}$ or by the security experiment. We remark that while values of $H$ on defined tuples might be known to the adversary, the adversary cannot (information-theoretically) guess a value $H(K, M)$ on an undefined tuple $(K, M)$ any better than with probability $2^{-k}$.

Consider an execution of $\mathcal{A}$ and consider all signing queries made by $\mathcal{A}$ (each signing query $M[1..n]$ is an $n$-tuple $(M_1, \ldots, M_n)$ of message symbols from $\{0, 1\}^k$). We define the following tree associated to the execution of $\mathcal{A}$. Its root contains a key $\text{Tag}[\varepsilon]$ computed in the security experiment and corresponds to the empty message. Nodes of the tree correspond to all prefixes of signing queries made by $\mathcal{A}$ in the security experiment and a node corresponding to $M[1..j] = (M_1, \ldots, M_j)$ contains the authentication tag $\text{Tag}[M[1..j]]$. Authentication tags for all the nodes in the tree are computed in the security experiment by oracle $\text{Extract}(\cdot)$. The adversary $\mathcal{A}$ receives authentication tags for all nodes corresponding to his signing queries. We call these nodes and all its descendants as revealed nodes. This tree also contains a forged message returned by $\mathcal{A}$ and all its ancestors. We recall that the forged message is fresh if the node corresponding to the forged message doesn’t have any of the revealed nodes as an ancestor. Figure 4.2 depicts an example of such a tree. $N$ is the bound on maximal number of nodes in the tree plus the maximal number of random oracle queries made by $\mathcal{A}$.

The proof proceeds as follows. First, we specify the event $\text{DISTINCT}$ and show that this event occurs with high probability. Then we argue that in the case the event $\text{DISTINCT}$ happens, the adversary has (information-theoretically) only a negligible chance in producing a valid forgery.
Figure 4.2 An exemplary tree constructed during the execution of $\mathcal{A}$. Grey nodes correspond to $\mathcal{A}$’s signing queries and a crossed node corresponds to the forged message returned by $\mathcal{A}$. All nodes in the tree contain authentication tags on the corresponding messages. In this example, $\mathcal{A}$ made signing queries $(A, E), (B, F), (B, G, I), (C)$ and $(C, H)$ and returned a forgery on message $(A, D)$. All the white nodes (including the crossed node) constitute the set $S$ of unrevealed nodes.

Let $\text{DISTINCT}$ denote the event (depending only on the random choice of $H$) that each value in an unrevealed node of the tree is distinct from all the other values in the tree and also distinct from any first components of random oracle queries to $H(\cdot, \cdot)$ made by $\mathcal{A}$. We claim that

$$\Pr[\neg \text{DISTINCT}] \leq \frac{N(N + 1)}{2^{k+1}},$$

the probability being taken over a random choice of $H$. Denote by $S$ a set of all unrevealed nodes in the tree. Let’s see how the set $S$ evolves over the execution of $\mathcal{A}$ and when the conditions of the event $\text{DISTINCT}$ get violated. We claim that at each moment of time $S$ satisfies to the following 2 conditions:

1. With high probability, each value in $S$ is distinct from all values in the tree as well
as from all first components of $\mathcal{A}$’s queries to $H(\cdot, \cdot)$.

2. If the first condition holds, values in all the nodes from $S$ are independent and uniformly distributed.

We show this recursively. Initially $S$ only contains a root of the tree, whose value is drawn randomly from $\{0, 1\}^k$, that is both of the conditions hold. When the new node is added to $S$, the value in that new node is $H(\sigma, M)$, which is an independent random value. It is not previously defined due to the first condition on $S$: $\sigma$ is different from all the first components of $H(\cdot, \cdot)$ queries by $\mathcal{A}$ and $(\sigma, M)$ is also different from all $H(\cdot, \cdot)$ calls made by the security experiment. This value matches values in other nodes in $S$ with probability $|S| \cdot 2^{-k}$. Next, when the adversary makes a query $(K, M)$ to $H(\cdot, \cdot)$, $K$ matches any of the values of nodes in $S$ with probability at most $|S| \cdot 2^{-k}$ (because of the second condition on $S$, values in $S$ are uniformly distributed). Since the total number of nodes in the tree plus the total number of $\mathcal{A}$’s oracle queries is upper-bounded by $N$ and at each addition of a node to $S$ as well as at each oracle query the probability of violating the first condition is at most $|S| \cdot 2^{-k}$, then the probability that it ever violated is at most

$$\frac{1}{2^k} + \cdots + \frac{N}{2^k} = \frac{N(N + 1)}{2^k}.$$

If the above 2 conditions on $S$ hold throughout the execution of $\mathcal{A}$, then the event DISTINCT also holds. Therefore

$$\Pr[\neg \text{DISTINCT}] \leq \frac{N(N + 1)}{2^{k+1}}.$$

Finally, we need to show that in the case that the event DISTINCT happens, the adversary wins the experiment only with negligible probability. Consider the execution of the adversary $\mathcal{A}$ and assume that the event DISTINCT happens. Let $M^*[1..n] = (M^*_1, \ldots, M^*_n)$ be a forged message returned by the adversary. Consider the tree and the set $S$ of unrevealed nodes corresponding to the execution of $\mathcal{A}$. In order for a forgery to be valid, a node corresponding to $M^*[1..n]$ must not be revealed, that is it must be in $S$. Denote by $\sigma$ a true authenticated tag for $M^*[1..n]$. From above, we know that $\sigma$ is
uniformly distributed and (information-theoretically) not known to the adversary. This implies that the adversary cannot guess \( \sigma \) any better than with probability \( 2^{-k} \), that is

\[
\Pr [ \mathcal{A} \text{ wins } | \text{DISTINCT} ] \leq \frac{1}{2^k}.
\]

Putting it all together, we have that

\[
\text{Adv}^{\text{aoma-uf-cma}}_{\text{AOMA1,A}}(k) \leq \Pr [ \neg \text{DISTINCT} ] + \Pr [ \mathcal{A} \text{ wins } | \text{DISTINCT} ] \\
\leq \frac{N(N + 1)}{2^{k+1}} + \frac{1}{2^k}.
\]

4.5 Applications

In this section, we describe two practical scenarios in which append-only signatures and message authentication schemes are directly applicable.

4.5.1 Wide-area Routing Protocol Security

An important application of AOS is in the construction of secure routing protocols for the Internet. The Border Gateway Protocol (BGP), which is the primary routing protocol used today in the Internet, has some well-known security weaknesses which require cryptographic solutions. While there have been many proposals for securing BGP in the past [43, 38, 60, 61], each must develop its own cryptographic constructions due to the lack of any primitive designed specifically for this application. In the discussion below, we briefly describe Internet routing and explain how our primitive is useful for ensuring one of the most important security requirements in BGP, namely path authenticity. Indeed, providing a sufficient cryptographic primitive for this problem led us to design AOS.

We begin with BGP, the Internet’s primary routing protocol, which is tasked with advertising paths from one network to all other networks. Each network, named by an
Autonomous System (AS) number, uses BGP to advertise the sets of IP addresses it is responsible for to its neighbor ASes. Each AS, upon receiving such advertisements, appends itself to the list of ASes on the forwarding path and repropagates the advertisement if it adheres to some local policies. When an AS receives two path advertisements for the same IP address space, it must make a decision as to which it wishes to use for its purposes and also to propagate to neighbors. Finally, once the set of routes have converged, these routes are used for packet forwarding; for each packet, the router looks up the destination IP address and forwards it to the neighbor AS as given by the BGP path advertisement.

Unfortunately, this path advertisement process also allows for any intermediate AS to hijack the process by changing advertisements arbitrarily. For example, if an AS truncates the AS path in an advertisement, then its neighbors will receive an advertisement shorter than the true path (typically causing them to prefer it). In the worst case, an AS can use this attack to convince its neighbors to forward all their traffic to it, which it could then modify or drop at will. (There are several other classes of attacks against BGP, but path modification and truncation are the most significant.)

Append-only Signatures can easily be applied to solve this problem as follows. Suppose that an AS $R_0$ wishes to announce routes for some IP prefix using the above path advertisement process. It first generates an AOS public-private key pair, distributes the public key AOS.pk throughout the network (this can be done with the help of a trusted authority that certifies public keys of ASes as in [43, 38, 61]) and to every neighboring AS $R_{i_1}$, sends the usual BGP information relating to the single-node path $(R_0)$ along with the AOS signature $\text{AOS.Append}(\text{AOS.pk}, \text{Sig}[\varepsilon], R_{i_1})$. In order to continue the advertisement process, $R_{i_1}$ sends to each of its own neighbors $R_{i_2}$ a BGP announcement containing the path $(R_0, R_{i_1})$ and the AOS signature $\text{AOS.Append}(\text{AOS.pk}, \text{Sig}[R_{i_1}], R_{i_2})$. In other words, $R_0$ appends the label of its neighbor $R_{i_1}$ into the AOS signature chain and $R_{i_1}$ further appends the label of $R_{i_2}$ into it$^3$. The advertisement

$^3$This may seem a bit unintuitive because each AS appends the “succeeding” AS’s identity, rather than its own, into the AOS signature. However, this is important to ensure security of the protocol; otherwise, a malicious AS can make a path arbitrarily long by appending random ASes into the path before it finally
process continues in this manner until all ASes in the network receive information about
a path to $R_0$. Each recipient can verify the validity of the announced path by verifying
the corresponding AOS signature using the public key $AOS.pk$. If the AOS scheme is
secure according to our definition (defn. 4.1.1), then all that a malicious AS can do
is append the label of one of its neighbors into the AOS signature chain (since each
neighbor $R_i$ can check that the AS it receives an advertisement from was the last to be
 appended before $R_i$).

In practice, the number of path advertisements received by any AS to a given
source AS $R_0$ is extremely small: as observed in real routing data [38], the odds that an
AS receives more than 15 path advertisements coming from the same source are about
1 in a 1000. This allows the use of efficient $m$-time signature schemes (as in Section
4.2.7), with $m$ equal to 15, in order to implement the AOS scheme in the above protocol
and obtain a reasonable level of security.

4.5.2 Secure Delegation of Resources

Fu et al. describe the SHARP system for distributed resource management, in
which users wish to share their resources with each other in a fully decentralized sys-
tem [28]. Central to this system is the notion of a claim—users are issued claims on
resources which they present upon resource use. These claims are signed such that they
can be verified to be valid by third parties. Furthermore, users can delegate their claims
to others, restricting them in the process. In this setting, the resource owner wishes to
place some restrictions on how her resources are delegated. Their setting allows for
a direct application of AOS. Quite simply, each resource provider, when initially issu-
ing a resource claim, appends to an AOS the amount of resources to be given. Upon
delegation, subsequent parties simply append to the signature what fraction of the exist-


4The security of this solution relies on the assumption that AS-to-AS links are authenticated in some
standard manner, for example using Message Authentication Codes (MACs) or existing infrastructure like
IPSec, as done in [38, 61]. Also, the AOS-based approach is not resilient to collusions between multiple
malicious ASes, as is the case with all proposals for securing BGP that we are aware of.
ing claim is to be delegated. Upon claiming resources, the holder of the sub-delegated claim cannot use more than the fraction of the original resources indicated by the AOS signature.

### 4.5.3 Authenticated Source-Routing

We also observe that append-only message authentication (AOMA) schemes have applications in wide-area source routing. In Platypus, an authenticated source-routing system, users send source-routed packets containing message authentication (MA) tags that are verified by routers before forwarding [55]. Users performing source routing and the routers providing this forwarding service share keys under which these MA tags are computed. In this context, users need the ability to delegate their source-routing ability to third parties, but do so in a way that restricts the flexibility of these parties. (In particular, third parties are restricted in the IP prefix to whom they can send packets.) In the system, the routers performing verification and the initial users share a key, but the routers are not involved in the delegation process. The result of the delegation process is a “restricted” tag that is used to derive a key under which MA of packets are computed. Since the verifier is a router, verification must be as fast as possible. In this context, AOMA schemes are directly applicable, as each party simply appends to the AOMA tag they received to restrict it further. We note that the mechanism used in the paper to authenticate source-routed packets, the “double-MAC trick”\(^5\), as it is known in the literature [2, 30, 51], is simply a two-round variant of AOMA scheme.

### 4.6 Final Remarks and Open Problems

#### 4.6.1 Finalization of AOS signature

An interesting feature of append-only signature schemes which might be needed by some applications is the ability to “finalize” the signature, that is, to modify the

---

\(^5\)Note that this use of the term “double-MAC” by several papers conflicts with terminology used by Petrank and Rackoff in a FIL-MAC to VIL-MAC construction for the CBC MAC [52].
signature of a message in the way which prohibits any further appending. The general solution to this problem is to use a special symbol $\Theta$ (from the message space) to denote the end of the message. When one wants to finalize the signature of some message, he should append $\Theta$ to the signature. Messages that contain symbol $\Theta$ in the middle of the message (not as the last symbol) are therefore considered to be invalid.

### 4.6.2 Restricted AOS

In AOS, anyone can append and verify signatures. In certain scenarios, however, one may want to restrict the ability to append messages to a limited group of users. Still, anyone should be able to verify the signatures. We call this extension of AOS Restricted Append-Only Signatures (RAOS).

Let $\mathcal{U}$ be a group of users allowed to perform the append operation. We assume that all members of the group $\mathcal{U}$ are given some key $K$ of an (symmetric) encryption scheme $\mathcal{ENC} = (ENC, DEC)$.

We modify a given AOS scheme to get a RAOS scheme as follows: Define the RAOS signature of a message $M[1..n] = (M_1, \ldots, M_n)$ as the tuple

$$\text{Sig}' = (\text{ENC}_K(\text{Sig}(M[1..n])), \text{Sig}(M_1, \ldots, M_n, \Theta)),$$

where $\Theta$ is the finalization symbol from the last paragraph. In order to append the message $M_{n+1}$ to a given RAOS signature on $M[1..n]$, a member of the group $\mathcal{U}$ decrypts the first part of the RAOS signature with her key $K$ to obtain $\text{Sig}(M[1..n])$. She then appends $M_{n+1}$ using the original AOS.Append algorithm. Finally, she outputs the new RAOS signature tuple by encrypting $\text{Sig}(M[1..n+1])$ and appending $\Theta$ to $\text{Sig}(M[1..n+1])$. Note that without knowledge of the key $K$, the AOS signature $\text{Sig}(M[1..n])$ remains secret and hence appending cannot be performed. Public verification is done by verifying if the second part of the RAOS signature is a valid AOS signature on the message $(M_1, \ldots, M_n, \Theta)$. 
4.6.3 Shorter AOS signatures

Given that wide-area routing protocols propagate a large number of messages, compact signatures are desirable. Thus we raise an open problem of whether it is possible to build an AOS scheme with constant signature length (in both message length and maximal message length). This problem is equivalent to building a HIBS scheme where secret keys of the users have constant length (in the depth of the given user in the hierarchy and in the maximal depth of the hierarchy).

So far the best we can get is the construction from Section 4.2.4 which provides an AOS scheme that can sign messages of length up to $n$ symbols with signatures of length $O(\sqrt{n})$. This construction translates to a HIBS scheme with $n$-level deep hierarchy, where the secret key of each user has length $O(\sqrt{n})$.

4.7 Acknowledgements

Parts of this chapter appeared in the paper


The dissertation author was the primary investigator and author of this paper.
Bibliography


