Automated Analysis and Synthesis of Authenticated Encryption Schemes

Viet Tung Hoang
University of Maryland
Georgetown University
tvhoang@umd.edu

Jonathan Katz
University of Maryland
jkatz@cs.umd.edu

Alex J. Malozemoff
University of Maryland
amaloz@cs.umd.edu

ABSTRACT

Authenticated encryption (AE) schemes are symmetric-key encryption schemes ensuring strong notions of confidentiality and integrity. Although various AE schemes are known, there remains significant interest in developing schemes that are more efficient, meet even stronger security notions (e.g., misuse-resistance), or satisfy certain non-cryptographic properties (e.g., being patent-free).

We present an *automated* approach for analyzing and synthesizing blockcipher-based AE schemes, significantly extending prior work by Malozemoff et al. (CSF 2014) who synthesize encryption schemes satisfying confidentiality only. Our main insight is to restrict attention to a certain class of schemes that is expressive enough to capture several known constructions yet also admits automated reasoning about security. We use our approach to generate thousands of AE schemes with provable security guarantees, both known (e.g., variants of OCB and CCM) and new. Implementing two of these new schemes, we find their performance competitive with state-of-the-art AE schemes.

1. INTRODUCTION

Historically, symmetric-key encryption schemes were designed only to ensure confidentiality. With the realization that practitioners were often (implicitly) assuming that such schemes also provided some form of integrity, however, researchers began explicit consideration and analysis of encryption schemes additionally satisfying that property [16, 7]. Since then, a tremendous amount of research has focused on the design of *authenticated encryption* (AE) schemes ensuring both confidentiality and integrity.

While a generic construction of an AE scheme based on any CPA-secure encryption scheme and message authentication code is possible [7], more efficient AE schemes can be devised. One example is OCB [25, 23, 17], which is online (i.e., requires only a single pass over the data), provably secure, and very fast. Unfortunately, due to patent restrictions, the scheme never gained widespread use. Other well-known AE schemes include CCM [10] and GCM [20]; however, these schemes are slower than OCB [17] and have other disadvantages as well.[1] Overall, the problem of designing AE schemes is still of interest, as evidenced by the ongoing CAESAR competition [9].

In designing efficient AE schemes, one might fail to realize opportunities to improve efficiency. For example, OCB was first introduced in 2001 [25] and its authors have been active in maintaining and optimizing the scheme, releasing OCB2 in 2004 [23] and OCB3 in 2011 [17]. However, recently, Minematsu [21] showed that a simple change to OCB's design allows one to use only the *forward direction* of the underlying blockcipher, saving chip area in hardware realizations.

In this work, we propose an *automated* approach for analyzing and synthesizing AE schemes. Our approach builds on and extends the work of Malozemoff et al. [19], who explored a similar goal but limited to encryption schemes achieving confidentiality only. At a high level, as in their work, we view an encryption scheme as being defined by a directed acyclic graph in which each node corresponds to an instruction (e.g., XORing two values) and is associated with an intermediate n-bit value. The graph defines how individual message blocks are processed; messages of arbitrary length are encrypted by iterating the computation defined by this graph over all the blocks of the message. (We actually consider processing *two* message blocks at a time, as this allows us to capture more AE schemes within our framework.) We develop a type system for the nodes of such graphs, and define constraints on how nodes can be typed based on their parents' types. We then show that any "well-typed" graph defines a secure AE scheme. This allows us to automatically *analyze* a given scheme by checking whether the graph defining the scheme can be properly typed. Building on this, we can *synthesize* schemes by enumerating over valid graphs and analyzing each one to see if it is secure. Through a generic transformation, our work can handle messages of arbitrary length, whereas the prior work of Malozemoff et al. is limited to messages whose length is a multiple of the block length.

Although the high-level structure of our approach is similar to that of Malozemoff et al., the technical details differ greatly due to the added challenge of handling integrity. (Indeed, this was left as an explicit open question in their work.) We were unable to *directly* extend their work to deal with integrity; instead, we modify their approach and consider

CCS'15, October 12–16, 2015, Denver, CO, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3832-5/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2810103.2813636.

[1]For example, GCM is fairly complex and has a problematic security proof [14], whereas CCM is not online and cannot pre-process associated data.

a restricted class of encryption schemes for which an automated analysis of integrity is tractable. Specifically, we focus on schemes constructed using *tweakable* blockciphers [18] in a particular way[2]. Several existing AE schemes satisfy our requirements, indicating that our framework is not overly restrictive. The abstraction from using a tweakable blockcipher also significantly reduces the size of the graphs that we have to enumerate, making the synthesis feasible. Despite this simplification, our graphs are a lot more complex than those that Malozemoff et al. consider. For example, in the prior work, it is relatively easy to tell if a scheme is decryptable, as there is only one path from a node representing a plaintext block to a node representing a ciphertext block. In our case, this no longer holds—the graphs of schemes like OTR [21] have multiple paths between such pairs of nodes, and we have to find a nontrivial algorithm to explicitly construct a graph of the encryption scheme, given a graph of the decryption scheme.

Using our approach, we are able to synthesize thousands of secure AE schemes, hundreds of which are "optimal" in the sense that they use only *one* tweakable blockcipher call per block, on par with OCB. These schemes are provably secure, as verified by our analysis tool, with *concrete* security bounds. In contrast, the prior work of Malozemoff et al. [19] only gives asymptotic analyses. We also employ a simple algorithm to find *fully parallelizable* constructions among the "optimal" schemes, and discover seventeen such schemes, five of which use the same number of instructions as OCB. We implement two of these schemes and find that the running times are comparable to those of OCB. Thus, these schemes may be of interest to practitioners looking for efficient, simple, and patent-free AE schemes. Finally, in the full version [13] of this work, we devise a method for automatically finding attacks on schemes that our approach cannot claim secure, and we find that most of those schemes indeed are susceptible to concrete attacks.

Related work. Recently there has been a growing interest in applying automated techniques to the analysis and design of cryptographic primitives. In the public-key setting, Barthe et al. [4] introduced an approach applicable to RSA-based encryption schemes. More recently, Tiwari et al. [26] developed a unified technique for synthesizing both RSA-based encryption schemes and modes of operation, among other cryptographic primitives. Other work has looked at automated analysis of assumptions in generic groups [5] with applications to automated synthesis of signature schemes having certain properties [5, 6]. Finally, Akinyele et al. [1, 2] developed tools for analyzing signature and encryption schemes to determine when (and how) known secure transformations can be applied.

2. PRELIMINARIES

Notation. Let $\mathbb{Z}$ denote the set of all integers, and let $\mathbb{N}$ denote the set of positive integers. Let $\{0,1\}^*$ denote the set of all binary strings, including the empty string. For a string M, let $|M|$ denote the length of M. For $M \in \{0,1\}^*$ and $1 \le i \le j \le |M|$, let $M[i]$ denote the i-th bit of M, and $M[i,j]$ the substring of M from the ith to the jth bit, inclusive. For two strings X and Y, we write XY or $X \,\|\, Y$ to denote the concatenation of X and Y.

We write $x \leftarrow\!\!\$\; S$ to denote uniform sampling of x from finite set S. For finite sets S_1, S_2, and random variables $X, Y \in S_1$, $Z \in S_2$, define $\|X - Y \mid Z\|$, the statistical distance between X and Y given Z, as

$$\frac{1}{2}\sum_{v\in S_1, z\in S_2} \Pr[Z=z]\cdot\Big|\Pr[X=v \mid Z=z] - \Pr[Y=v \mid Z=z]\Big|.$$

Games. We use the code-based, game-playing framework of Bellare and Rogaway [8]. Due to lack of space, we assume the reader is familiar with this framework.

Tweakable blockciphers [18]. Let $n \in \mathbb{N}$. A tweakable blockcipher on n-bit strings with tweak space $\mathcal{T}$ and key space $\mathcal{K}$ is a map $E : \mathcal{K} \times \mathcal{T} \times \{0,1\}^n \to \{0,1\}^n$ such that $E_K(T, \cdot)$ is a permutation on $\{0,1\}^n$ for any $K \in \mathcal{K}$ and $T \in \mathcal{T}$. Let E^{-1} denote the inverse of E, meaning $E_K^{-1}(T, E_K(T,x)) = x$ for $K \in \mathcal{K}$, $T \in \mathcal{T}$, and $x \in \{0,1\}^n$. For brevity we sometimes write $E_K^T(x)$ for $E_K(T,x)$. Define the *strong tweakable-PRP advantage* of an adversary $\mathcal{A}$ against E as

$$\mathbf{Adv}_E^{\pm\widetilde{\mathrm{prp}}}(\mathcal{A}) = \Big|\Pr[K \leftarrow\!\!\$\; \mathcal{K}\colon\ \mathcal{A}^{E_K(\cdot,\cdot), E_K^{-1}(\cdot,\cdot)} \Rightarrow 1] - \Pr[\pi \leftarrow\!\!\$\; \mathrm{Perm}(\mathcal{T}, n)\colon\ \mathcal{A}^{\pi(\cdot,\cdot),\pi^{-1}(\cdot,\cdot)} \Rightarrow 1]\Big|,$$

where $\mathrm{Perm}(\mathcal{T}, n)$ is the set of all $\mathcal{T}$-indexed families of permutations on $\{0,1\}^n$. (I.e., $\mathrm{Perm}(\mathcal{T}, n)$ is the set of all functions $\pi : \mathcal{T} \times \{0,1\}^n \to \{0,1\}^n$ with the property that for each $T \in \mathcal{T}$, the reduced function $\pi(T, \cdot)$ is a permutation on $\{0,1\}^n$.) If the adversary is prohibited from making queries to the second oracle, we drop the word "strong" and write $\mathbf{Adv}_E^{\widetilde{\mathrm{prp}}}(\mathcal{A})$ instead.

Authenticated encryption. Rather than view encryption schemes as being randomized or stateful, we follow Rogaway [24] in viewing them as deterministic transformations that take as input a message along with some associated data (which need not be kept secret) as well as a user-supplied nonce. Security is then required to hold as long as the same nonce is never used twice.

Formally, an authenticated encryption (AE) scheme [7, 16, 22, 24] is a tuple $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ with key space $\mathcal{K}$, nonce space $\mathcal{N}$, associated data space $\mathcal{A}$, message space $\mathcal{M}$, and tag length $\tau \in \mathbb{N}$. Both algorithms $\mathcal{E}$ and $\mathcal{D}$ are deterministic. The encryption algorithm $\mathcal{E}$ maps an input tuple $(K, N, A, M) \in \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M}$ to a ciphertext $C \in \{0,1\}^*$. Decryption $\mathcal{D}$ reverses encryption, mapping an input tuple $(K, N, A, C) \in \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \{0,1\}^*$ to either a message $M \in \mathcal{M}$ or a distinguished error symbol $\bot$. The correctness requirement demands that $\mathcal{D}_K^{N,A}(\mathcal{E}_K^{N,A}(M)) = M$ for every $(K, N, A, M) \in \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M}$.

We define the privacy advantage of an adversary $\mathcal{A}$ against an AE scheme Π as $\mathbf{Adv}_\Pi^{\mathrm{priv}}(\mathcal{A}) =$

$$\Big|\Pr[K \leftarrow\!\!\$\; \mathcal{K}\colon\ \mathcal{A}^{\mathcal{E}_K(\cdot,\cdot,\cdot)} \Rightarrow 1] - \Pr[\mathcal{A}^{\$(\cdot,\cdot,\cdot)} \Rightarrow 1]\Big|,$$

where $\$(\cdot,\cdot,\cdot)$ is an oracle that, on any input (N, A, M), outputs a fresh, uniform $(|M| + \tau)$-bit answer. We require here that the adversary never uses the same nonce twice as input to its oracle. Informally, a scheme satisfies privacy if the privacy advantage of any efficient adversary is small.

[2]Roughly, a tweakable blockcipher accepts a "tweak" in addition to a key and a regular input; for a fixed key, different tweaks should produce "independent-looking" permutations. See the following section for a formal definition.

Nonces used by the honest party during encryption need only be unique[3], not uniform.

For authenticity, the adversary is again given access to an encryption oracle $\mathcal{E}_K(\cdot,\cdot,\cdot)$, and as before must not use the same nonce twice. We say that $\mathcal{A}$ *outputs a forgery* if it outputs (N, A, C) such that $\mathcal{D}_K(N, A, C) \neq \perp$ and C was not the result of a prior oracle query $\mathcal{E}_K(N, A, M)$ for some message M. We define the authenticity advantage of $\mathcal{A}$ as $\mathbf{Adv}^{\text{auth}}_{\Pi}(\mathcal{A}) = \Pr[K \leftarrow\$\ \mathcal{K} \colon \mathcal{A}^{\mathcal{E}_K(\cdot,\cdot,\cdot)}$ outputs a forgery]. Informally, a scheme satisfies authenticity if the authenticity advantage of any efficient adversary is small.

3. AUTOMATED SECURITY ANALYSIS

We now describe our approach to the automated analysis of AE schemes constructed from tweakable blockciphers following a particular template (cf. Section 3.1). Although this template does not capture all known AE schemes, it is expressive enough to include simplified variants of, e.g., OCB [23], XCBC [12], COPA [3], OTR [21], and CCM [10].[4]

As discussed in the Introduction, we view an encryption scheme as being defined by a directed acyclic graph in which each node is associated with an instruction and carries an n-bit intermediate value. In Section 3.2 we describe a type system for the nodes of such graphs, and show how to use these types for reasoning about properties of the intermediate values that those nodes carry. Then, in Section 3.3, we show how this reasoning enables us to automatically verify whether an AE scheme, given by its graph representation, satisfies privacy and authenticity.

3.1 A Template for AE Schemes

Fix associated data space $\mathcal{A}$, and let $\mathcal{N} = \{0,1\}^n$. Let $\mathcal{T} = \mathcal{N} \times \mathcal{A} \times \mathbb{Z}$ and let $E : \mathcal{K} \times \mathcal{T} \times \{0,1\}^n \to \{0,1\}^n$ be a tweakable blockcipher.We consider AE schemes $\Pi[E] = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ that use E as an oracle. The schemes we consider have message space[5] $\mathcal{M} = (\{0,1\}^{2n})^*$ and are built from algorithms $(\mathsf{Enc}, \mathsf{Dec}, \mathsf{Tag})$ having the following form:

- Enc^{E_K} takes as input tweak $T = (N, A, v) \in \mathcal{T}$, an initial state $X \in \{0,1\}^{2n}$, and a (double-length) message block $M \in \{0,1\}^{2n}$. It outputs a (double-length) ciphertext block $C \in \{0,1\}^{2n}$ and final state $Y \in \{0,1\}^{2n}$. This algorithm makes a fixed number of queries to E_K, denoted by $\mathbf{Cost}(\Pi)$, and we require that the tweak in the ith such query is $(N, A, v+i-1)$.

```
$\underline{\mathcal{E}_K(N, A, M)}$
$X := 0^{2n};\ v := 1;\ M_1 \cdots M_{2m} := M$    // $|M_i| = n$
for $i = 1$ to $m$ do
    $T := (N, A, v)$
    $(Y, C_{2i-1}C_{2i}) := \mathsf{Enc}^{E_K}(T, X, M_{2i-1}M_{2i})$    // $|C_j| = n$
    $v := v + \mathbf{Cost}(\Pi);\ X := Y$
$T := (N, A, 1-v);\ V := \mathsf{Tag}^{E_K}(T, X)$
return $C_1 \cdots C_{2m} \parallel V[1, \tau]$

$\underline{\mathcal{D}_K(N, A, C)}$
if $|C| \not\equiv \tau \pmod{2n}$ then return $\perp$
$C_1 \cdots C_{2m} \parallel tag := C$   //  $|C_i| = n$ and $|tag| = \tau$
$X := 0^{2n};\ v := 1$
for $i = 1$ to $m$ do
    $T := (N, A, v)$
    $(Y, M_{2i-1}M_{2i}) := \mathsf{Dec}^{E_K, E_K^{-1}}(T, X, C_{2i-1}C_{2i})$
    $v := v + \mathbf{Cost}(\Pi);\ X := Y$
$T := (N, A, 1-v);\ V := \mathsf{Tag}^{E_K}(T, X)$
if $tag \neq V[1, \tau]$ then return $\perp$
else return $M_1 \cdots M_{2m}$
```

Figure 3.1: Code of an AE scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ following our template. The scheme is based on a tweakable blockcipher E and a triple of deterministic algorithms $(\mathsf{Enc}, \mathsf{Dec}, \mathsf{Tag})$.

- $\mathsf{Dec}^{E_K, E_K^{-1}}$ "inverts" algorithm Enc in the following sense: if $\mathsf{Enc}^{E_K}(T, X, M) = (Y, C)$ then it holds that $\mathsf{Dec}^{E_K, E_K^{-1}}(T, X, C) = (Y, M)$.
- Tag^{E_K} takes as input tweak $T \in \mathcal{T}$ and initial state $X \in \{0,1\}^{2n}$, and produces a tag $V \in \{0,1\}^n$. It makes a single query to E_K using tweak T.

The encryption/decryption algorithms $(\mathcal{E}, \mathcal{D})$ of Π are then defined as in Figure 3.1, where we require $\tau \leq n$. Roughly, to encrypt a message $M = M_1 M_2 \cdots M_{2m}$ using nonce N and associated data A, set the initial state $X = 0^{2n}$ and set $T = (N, A, 1)$. Then, iteratively process two message blocks at a time using Enc, each time updating the initial state and outputting the next two ciphertext blocks. After processing the entire message, Tag is used to compute a tag based on the final state output by Enc and a designated tweak that depends on the message length; the (truncated) tag is appended to the ciphertext.

Graph representation. As in the work of Malozemoff et al. [19], we represent algorithms Enc, Dec, and Tag as directed acyclic graphs, where each node is associated with an instruction and carries an n-bit value. The n-bit value on each node is determined by applying the instruction at that node to the values at the parent nodes. In the next section we introduce a system for "typing" the nodes of such graphs; our main theorem states that AE schemes built from Enc, Dec, and Tag algorithms whose graphs can be correctly typed are secure.

The main instructions we support are `XOR`, which computes the XOR of two n-bit strings, and `TBC`, which invokes the tweakable blockcipher or its inverse. We also have an instruction `DUP` that duplicates a value. Nodes corresponding to input blocks are labeled `IN`, those corresponding to output blocks are labeled `OUT`, those corresponding to the initial state are labeled `INI`, and those corresponding to the final state are labeled `FIN`. These labels, along with their in-/out-degree, are summarized for convenience next:

[3]The requirement that nonces be unique is necessary, since repeating (N, A, M) will repeat the corresponding ciphertext. For real schemes such as OCB, reusing a nonce is devastating, damaging the privacy and authenticity of not just past queries, but also future ones. It is the responsibility of the implementation to ensure that nonces are unique.

[4]For efficiency, the real-world variants are often built directly from a blockcipher instead of a tweakable one, and employ a scheme-specific way to handle fragmentary data. The real-world CCM is not online due to its treatment of fragmentary data, whereas our variant is online. OCB is built from a tweakable blockcipher, but the tweaks are (N, i) instead of (N, A, i). To handle associated data, OCB employs an XOR-universal hash (based on a tweakable blockcipher), and XORs the hash image to the tag.

[5]Messages of arbitrary length can be handled by naive padding in the usual way. In the full version [13] we describe a more efficient approach for handling messages of arbitrary length.

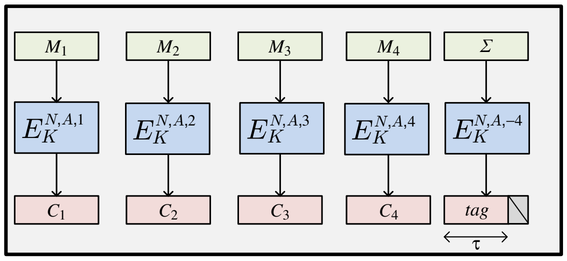

Figure 3.2: The OCB scheme illustrated for a four-block message $M_1, \ldots, M_4$, where Σ is the checksum $M_1 \oplus \cdots \oplus M_4$.

```
$\mathsf{Enc}^{E_K}(T, X, M_1, M_2)$
$(N, A, v) := T$
$X_1, X_2 := X$   // $|X_i| = n$
$C_1 := E_K(T, M_1)$
$T := (N, A, v + 1)$
$C_2 := E_K(T, M_2)$
$Y := (X_1 \oplus M_1 \oplus M_2) \| X_2$
return $(Y, C_1, C_2)$
```

```
$\mathsf{Dec}^{E_K, E_K^{-1}}(T, X, C_1, C_2)$
$(N, A, v) := T$
$X_1, X_2 := X$   // $|X_i| = n$
$M_1 := E_k^{-1}(T, C_1)$
$T := (N, A, v + 1)$
$M_2 := E_K^{-1}(T, C_2)$
$Y := (X_1 \oplus M_1 \oplus M_2) \| X_2$
return $(Y, M_1, M_2)$
```

```
$\mathsf{Tag}^{E_K}(T, X)$
$X_1, X_2 := X$; $V := E_K(T, X_1)$; return $V$
```

Figure 3.3: The algorithms $(\mathsf{Enc}, \mathsf{Dec}, \mathsf{Tag})$ corresponding to OCB. We have $\mathbf{Cost}(\mathrm{OCB}) = 2$.

Name	In-deg	Out-deg	Meaning
`IN`	0	1	Input block
`INI`	0	1	Initial state
`FIN`	1	0	Final state
`OUT`	1	0	Output block
`DUP`	1	2	Duplicate
`XOR`	2	1	XOR operation
`TBC`	1	1	Tweakable blockcipher

Figure 3.2 illustrates the OCB scheme [23], Figure 3.3 shows the corresponding Enc, Dec, and Tag algorithms, and Figure 3.4 shows the corresponding graphs. (In OCB, only the first n bits of the state are used, so we treat the state as an element of $\{0,1\}^n$.) Note that Figure 3.4 is informal and omits information needed to fully specify OCB; see next for formal details of how graphs are specified.

Formally, we denote a graph G by a tuple (d, r, F, P, L), where $d \in \{2, 4\}$ is the total number of IN and INI nodes (Enc and Dec graphs have $d = 4$; Tag graphs have $d = 2$), and $r \in \mathbb{N}$ is the total number of nodes. Each node in the graph is numbered from 1 to r, and we require that if node i is a parent of node j then $i < j$. (This ensures that G is acyclic.) Let *Nodes* denote the power set of $\{1, \ldots, r\}$, and let $\mathit{Inst} = \{\mathtt{IN}, \ldots, \mathtt{TBC}\}$ be the set of instructions. Then $F : \{1, \ldots, r\} \to \mathit{Inst}$ gives the instruction of each node and $P : \{1, \ldots, r\} \to \mathit{Nodes}$ gives the set of parents for each node. We require that $F(1) = F(2) = \mathtt{INI}$ and $F(r) = \mathtt{OUT}$. For Enc and Dec graphs, we additionally require that $F(3) = F(4) = \mathtt{IN}$, $F(r-2) = F(r-3) = \mathtt{FIN}$, and $F(r-1) = \mathtt{OUT}$.

For Enc and Dec graphs, let $S \subset \{1, \ldots, r\}$ be the set of all nodes corresponding to a TBC instruction. Function $L : S \to \mathbb{Z}$ specifies, for each such node i, whether the tweakable blockcipher is computed in the forward direction (if $L(i) \geq 0$) or the reverse direction (if $L(i) < 0$) at that

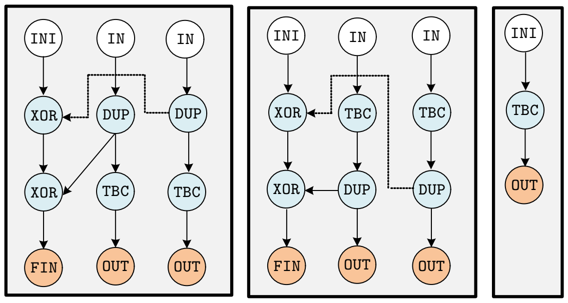

Figure 3.4: Graph representations for algorithms Enc (left), Dec (middle), and Tag (right) of OCB.

```
proc $\mathsf{Eval}^{E_K, E_K^{-1}}(G, T, Z_1, \ldots, Z_d)$
$(d, r, F, P, L) := G$
for $i = d + 1$ to $r$ do
  if $F(i) \in \{\mathtt{DUP}, \mathtt{OUT}, \mathtt{FIN}\}$ then $\{p\} := P(i)$; $Z_i := Z_p$
  elseif $F(i) = \mathtt{XOR}$ then $\{p_1, p_2\} := P(i)$; $Z_i := Z_{p_1} \oplus Z_{p_2}$
  else   // $F(i) = \mathtt{TBC}$
    $\ell := L(i)$; $(N, A, v) := T$
    $T^* := (N, A, v + |\ell|)$; $\{p\} := P(i)$
    if $\ell > 0$ then $Z_i := E_K(T^*, Z_p)$ else $Z_i := E_K^{-1}(T^*, Z_p)$
return $(Z_1, \ldots, Z_r)$
```

Figure 3.5: Procedure to compute the value Z_i of each node i in a graph G, given input $Z_1, \ldots, Z_d$ and tweak T.

node.[6] (Note that $L(i) \geq 0$ for Enc graphs.) Moreover, $|L(i)|$ determines the tweak at node i; i.e., on input tweak (N, A, v), the tweak at node i is $(N, A, v + |L(i)|)$.

Let $G^- = (d, r, F, P)$ denote the *unlabeled graph* corresponding to a graph G. In Section 3.2, we introduce a type system and show that one can reason about the security of an AE scheme by evaluating the scheme's unlabeled graphs.

Fix some graph G, tweakable blockcipher E, and key K. Given a tweak T and n-bit values for all INI/IN nodes in G, we can naturally define an n-bit value Z_i associated with each node i in the graph. We describe this formally as procedure Eval in Figure 3.5, which shows how to compute Z_i given values $Z_1, \ldots, Z_d \in \{0,1\}^n$ and tweak T.

3.2 A Type System for AE Schemes

Let $\mathit{Types} = \{\$, \bot, 0, 1\}$ be a set of "types" we can assign to nodes. Intuitively, '\$' indicates a node whose output value is (pseudo)random (when the key K for E is random and secret), whereas '$\bot$' indicates a node whose output value is arbitrary (i.e., potentially controlled by an attacker). Looking ahead, types '0' and '1' will be used to compare values on the same node in two different decryption queries using the same nonce and associated data; '0' means the corresponding values are the same, and '1' means they are different.

In Figure 3.6, we define a deterministic procedure Map that takes as input an unlabeled graph G^-, pre-assigned types $\mathit{type}_1, \ldots, \mathit{type}_d \in \mathit{Types}$ for all the INI/IN nodes, and a boolean flag *rand*, and returns a map R that associates each node i with a pair $(\mathit{type}_i, \mathit{ctr}_i) \in \mathit{Types} \times \mathbb{N}$. The procedure Map traverses the graph in topological or-

[6]While it might be conceptually simpler to use two different instructions for E_K and E_K^{-1}, instead of just a single TBC instruction with positive/negative labels, our approach is an optimization that prunes the search space when synthesizing schemes (cf. Section 4).

```
proc Map($G^-$, $type_1, \ldots, type_d$, $rand$)
$(d, r, F, P) := G^-$; $maxCtr := 0$
for $i = 1$ to $d$ do
  if $type_i = \$$ then $R(i) := (\$, 1)$; $maxCtr := 1$
  else $R(i) := (type_i, 0)$
for $i = d + 1$ to $r$ do
  if $F(i) \in \{\texttt{FIN}, \texttt{OUT}, \texttt{DUP}\}$ then
    $\{p\} := P(i)$; $R(i) := R(p)$
  elseif $F(i) = \texttt{TBC}$ then
    $\{p\} := P(i)$; $(x, ctr) := R(p)$
    if $x \in \{1, \$\}$ or $(rand = \mathsf{true})$ then
      $maxCtr := maxCtr + 1$; $R(i) := (\$, maxCtr)$
    else $R(i) := (x, ctr)$
  else   // $F(i) = \texttt{XOR}$
    $\{p_1, p_2\} := P(i)$; $(x, ctr) := R(p_1)$; $(y, ctr') := R(p_2)$
     // Assume that $ctr \geq ctr'$
    if $(x, y) \in \{(0, 0), (0, 1), (1, 0)\}$ then $R(i) := (x \oplus y, ctr)$
    elseif $x = \$$ and $ctr > ctr'$ then $R(i) := (\$, ctr)$
    else $R(i) := (\perp, ctr)$
return $R$
```

Figure 3.6: A procedure for generating a mapping $R : \{1, \ldots, r\} \to Types \times \mathbb{N}$ for a given unlabeled graph.

der and assigns types to each node of the graph based on the instruction associated with that node and the types of its parents. These types are used for probabilistic reasoning about the underlying n-bit values on that node; e.g., we show that if a node has type \$ then the n-bit value of that node is (pseudo)random. The *ctr* values are used as "timestamps" for values output by TBC nodes in order to determine independence among values of type \$. Finally, the *rand* flag denotes whether the nonce/associated data are fresh.

For FIN, OUT, and DUP nodes, Map simply propagates the type of the parent node. For TBC nodes, if the nonce or input is fresh then the output is (pseudo)random and independent of any prior random values, and so the node gets type \$; otherwise, we propagate the type of the parent node.

For XOR nodes, we have several cases. If the two input nodes x and y are typed $type_x$ and $type_y$, respectively, with $(type_x, type_y) \in \{(0, 0), (0, 1), (1, 0)\}$, then we type the XOR node as $type_x \oplus type_y$. We briefly explain this reasoning. The fact that $type_x, type_y \in \{0, 1\}$ means there is a prior query using the same nonce and associated data. If the two parents have type 0, indicating that the values computed at those nodes are equal in the two queries, then clearly the value computed at the XOR node is also equal in the two queries, and thus that node gets type 0. On the other hand, if one parent is typed 0 and the other is typed 1, then the value computed at the XOR node will be different from the corresponding value in the prior query, and thus the XOR node is assigned type 1. If $(type_x, type_y) = (1, 1)$ then we cannot say anything definitive and thus Map assigns type $\perp$ to the XOR node. Finally, suppose input node x has type \$. Here we utilize the *ctr* values. If the *ctr* value at x is different from the *ctr* value of y, then the (random) value of x is independent of the value of y, and hence we assign the XOR node type \$.

In the next two lemmas we show how determining the types for an *unlabeled* graph can be used to reason about the values that one obtains when evaluating the *labeled* graph. We first show that all values typed \$ by Map (when inputs are typed $\perp$ and hence may be under arbitrary control of the adversary) are indeed random when computed using Eval and a truly random tweakable permutation.

LEMMA 3.1. *Let $G = (d, r, F, P, L)$ and let $n \geq 1$ be an integer. Set $R := \mathsf{Map}(G^-, \perp, \ldots, \perp, \mathsf{true})$. Fix arbitrary $Z_1, \ldots, Z_d \in \{0, 1\}^n$ and $T \in \mathcal{T}$, and consider the following probabilistic experiment:*

1. *Choose $f \leftarrow\!\!\$\ \mathrm{Perm}(\mathcal{T}, n)$.*
2. *Run $(Z_1, \ldots, Z_r) := \mathsf{Eval}^{f, f^{-1}}(G, T, Z_1, \ldots, Z_d)$.*

Then for any j with $R(j) = (\$, ctr_j)$, the random variable Z_j is uniform and independent of $\{Z_i \mid ctr_i < ctr_j\}$.

PROOF. First note that for any node i and its parent p, we have $ctr_i \geq ctr_p$. Thus, there is a topological ordering $s_1, \ldots, s_r$ of the nodes such that the sequence $ctr_{s_1}, \ldots, ctr_{s_r}$ is non-decreasing, and $s_i = i$ for $i \leq d$. Write $i \prec j$ if node i precedes node j in this topological order. We prove by induction (with respect to $\prec$) that for all j we have (i) $type_j \in \{\perp, \$\}$ and (ii) if $type_j = \$$ then Z_j is uniform and independent of $\{Z_i \mid ctr_i < ctr_j\}$. Note that these claims are trivially true when $j \leq d$, because then $type_j = \perp$.

Suppose both claims hold for all $i \prec j$. If $F(j) \in \{\texttt{FIN}, \texttt{OUT}, \texttt{DUP}\}$ then let $p \prec j$ be the parent of j. Since $(type_j, ctr_j) = (type_p, ctr_p)$, the claims follow for j. If $F(j) = \texttt{TBC}$ then $type_j = \$$, proving claim (i). Since $f \leftarrow\!\!\$\ \mathrm{Perm}(\mathcal{T}, n)$, and Eval never repeats a tweak in querying f, we see that random variable Z_j is uniform and independent of $\{Z_i \mid ctr_i < ctr_j\}$, justifying claim (ii). Finally, say $F(j) = \texttt{XOR}$. Let i and t be the parents of j, and assume $t \prec i$. Then $type_t, type_i \in \{\perp, \$\}$, and thus so is $type_j$, proving claim (i). For claim (ii), note that $type_j = \$$ only if $type_i = \$$ and $ctr_i > ctr_t$. Since $ctr_j = ctr_i$, the claim follows. □

The next lemma proves a similar property for *pairs* of queries. Consider the query $(Y_1, \ldots, Y_r) := \mathsf{Eval}^{f, f^{-1}}(G, T, Y_1, \ldots, Y_d)$ followed by query $(Z_1, \ldots, Z_r) := \mathsf{Eval}^{f, f^{-1}}(G, T, Z_1, \ldots, Z_d)$, where each Z_i is either chosen equal to Y_i (and thus $type_i = 0$), distinct from Y_i (and thus $type_i = 1$), or uniformly (and thus $type_i = \$$). We show that for all nodes j of type \$ assigned by $\mathsf{Map}(G^-, type_1, \ldots, type_d, \mathsf{false})$, the statistical difference between Z_j and uniform is small, even conditioned on all the $\{Y_i\}$.

LEMMA 3.2. *Let $G = (d, r, F, P, L)$ and let $n \geq 1$ be an integer. Fix arbitrary $Y_1, \ldots, Y_r \in \{0, 1\}^n$ and $T \in \mathcal{T}$ such that the set $S = \{f \in \mathrm{Perm}(\mathcal{T}, n) \mid (Y_1, \ldots, Y_r) = \mathsf{Eval}^{f, f^{-1}}(G, T, Y_1, \ldots, Y_d)\}$ is non-empty. For each $i \leq d$, choose Z_i and $type_i$ in one of the following ways: (i) $Z_i = Y_i$ and $type_i = 0$, (ii) $Z_i \neq Y_i$ and $type_i = 1$, (iii) $Z_i \leftarrow\!\!\$\ \{0, 1\}^n$ and $type_i = \$$. Let $R = \mathsf{Map}(G^-, type_1, \ldots, type_d, \mathsf{false})$. Consider the following probabilistic experiment:*

1. *Choose $f \leftarrow\!\!\$\ S$.*
2. *Run $(Z_1, \ldots, Z_r) := \mathsf{Eval}^{f, f^{-1}}(G, T, Z_1, \ldots, Z_d)$.*

Then for any j with $R(j) = (\$, ctr_j)$, the statistical difference between the random variable Z_j and uniform, conditioned on $\{Z_i \mid ctr_i < ctr_j\}$ and all the $\{Y_i\}$, is at most $2 \cdot ctr_j / 2^n$.

PROOF. As in the previous lemma, there is a topological ordering $s_1, \ldots, s_r$ of the nodes such that $ctr_{s_1}, \ldots, ctr_{s_r}$ is non-decreasing and $s_i = i$ for $i \leq d$. Write $i \prec j$ if i precedes j in this topological order. We prove by induction (with

```
proc Priv($G_1^-, G_2^-$)
  // $G_1^-$ and $G_2^-$ are unlabeled graphs of Enc and Tag, respectively.
01 $(d_1, r_1, F_1, P_1) := G_1^-$; $(d_2, r_2, F_2, P_2) := G_2^-$
  // Check that output of Tag is random
02 $R := \mathsf{Map}(G_2^-, \bot, \bot, \mathsf{true})$; $(type, ctr) := R(r_2)$
03 if $type \neq \$$ then return false
  // Check that output blocks of Enc are random and independent
04 $R := \mathsf{Map}(G_1^-, \bot, \bot, \bot, \bot, \mathsf{true})$
05 $(type_1, ctr_1) := R(r_1 - 1)$; $(type_2, ctr_2) := R(r_1)$
06 return $((type_1 = \$) \wedge (type_2 = \$) \wedge (ctr_1 \neq ctr_2))$

proc Auth($G_1^-, G_2^-$)
  // $G_1^-$ and $G_2^-$ are unlabeled graphs of Dec and Tag, respectively.
11 $(d_1, r_1, F_1, P_1) := G_1^-$; $(d_2, r_2, F_2, P_2) := G_2^-$
  // Check that output of Tag is random when the nonce/associated data are fresh
12 $R := \mathsf{Map}(G_2^-, \bot, \bot, \mathsf{true})$; $(type, ctr) := R(r_2)$
13 if $type \neq \$$ then return false
  // Check that if there are two executions of Dec with the same initial state
  // but different input blocks, then the first half of the final state is random
14 for $(x, y) \in \{(0,1), (1,0), (1,1)\}$ do
15   $R := \mathsf{Map}(G_1^-, 0, 0, x, y, \mathsf{false})$; $(type, ctr) := R(r_1 - 3)$
16   if $type \neq \$$ then return false
  // Check that if the first half of the initial state input to Dec is random,
  // then the first half of the final state output by Dec is random
17 for $x, y, z \in \{0,1\}$ do
18   $R := \mathsf{Map}(G_1^-, \$, x, y, z, \mathsf{false})$; $(type, ctr) := R(r_1 - 3)$
19   if $type \neq \$$ then return false
  // Check that if there are two executions of Tag in which the first halves of the
  // initial states are different, then the resulting tags are random and independent
20 for $x \in \{0,1\}$ do
21   $R := \mathsf{Map}(G_2^-, 1, x, \mathsf{false})$; $(type, ctr) := R(r_2)$
22   if $type \neq \$$ then return false
23 return true
```

Figure 3.7: Tests to determine if a scheme Π satisfies privacy and authenticity, respectively.

respect to $\prec$) that for all j: (i) if $type_j = 0$ then $Z_j = Y_j$, (ii) if $type_j = 1$ then $Z_j \neq Y_j$, and (iii) if $type_j = \$$ then the statement of the lemma holds. These claims all trivially hold when $j \leq d$.

Suppose that all three of the claims hold for all $i \prec j$. If $F(j) \in \{\mathtt{FIN}, \mathtt{OUT}, \mathtt{DUP}\}$ then let $p \prec j$ be the parent of j. Since $(type_j, ctr_j) = (type_p, ctr_p)$, the claims follow easily in this case. If $F(j) = \mathtt{XOR}$, let $t, i \prec j$ be the parents of j, and assume $t \prec i$. Note that $type_j \in \{0,1\}$ only if $type_i, type_t \in \{0,1\}$ and at most one of these values is 1, in which case the claims all hold. On the other hand, $type_j = \$$ only if $type_i = \$$ and $ctr_i > ctr_t$, in which case the claims also follow. Finally, if $F(j) = \mathtt{TBC}$ then let i be the parent of j. Let $\ell = L(j)$, and let $T = (N, A, v)$. Then $Y_j = f(T^*, Y_p)$ and $Z_j = f(T^*, Z_p)$, where $T^* = (N, A, v + |\ell|)$. Consider the following cases:

Case 1. $type_i \in \{0, \bot\}$. Then $(type_j, ctr_j) = (type_i, ctr_i)$ and the claims follow.

Case 2. $type_i = 1$. Then $type_j = \$$ and $ctr_j \geq 1$. First, since $ctr_t \geq ctr_j$ when t is a descendant of j, we see that no node t with $ctr_t < ctr_j$ is a descendant of j. Next, since $Z_i \neq Y_i$ and we use a different tweak for each TBC node, $Z_j \leftarrow\$ \{0,1\}^n \backslash \{Y_j\}$ is independent of $\{Y_t \mid t \leq r\}$ and $\{Z_t \mid ctr_t < ctr_j\}$.

Case 3. $type_i = \$$. Then $type_j = \$$. By the induction hypothesis, Z_i is $(2ctr_i/2^n)$-close to uniform (even conditioned on $\{Y_t \mid t \leq r\}$ and $\{Z_t \mid ctr_t < ctr_i\}$). If $Z_i \neq Y_i$, which occurs except with probability at most $(2ctr_i + 1)/2^n$, then Z_j is 2^{-n}-close to uniform (even conditioned on all the $\{Y_t\}$ values and $\{Z_t \mid ctr_t < ctr_j\}$). Hence, overall, Z_j is $(2ctr_i + 2)/2^n$-close to uniform (conditioned on $\{Y_t \mid t \leq r\}$ and $\{Z_t \mid ctr_t < ctr_j\}$) and the statement of the lemma follows since we have $ctr_j \geq ctr_i + 1$. □

3.3 Verifying Privacy and Authenticity

We use Lemmas 3.1 and 3.2 to automatically check if a candidate AE scheme is secure in the sense of both privacy and authenticity. Specifically, Figure 3.7 shows procedures Priv and Auth to check for privacy and authenticity, respectively, of an AE scheme Π.

Intuitively, for privacy we verify that the tag and all the ciphertext blocks output by the scheme are random and independent (namely, have type \$ and distinct counter values) even when the inputs—that is, the message blocks—are controlled by the adversary (namely, have type $\bot$). We remark that the values of *ctr* assigned to nodes by the map R output by Map depend on the topological order in which Map traverses the input graph; see Figure 3.8 for an example. Thus, there *are* schemes which, depending on the order in which the graph is traversed, are accepted or (incorrectly) rejected by Priv (due to the *ctr* values for the OUT nodes being equal). This shows that the test is *sound* but not *complete*.[7]

[7]In the full version [13] we describe a technique for generating attacks given a scheme which fails the tests of Figure 3.7. Looking ahead, we find only a handful of schemes which we

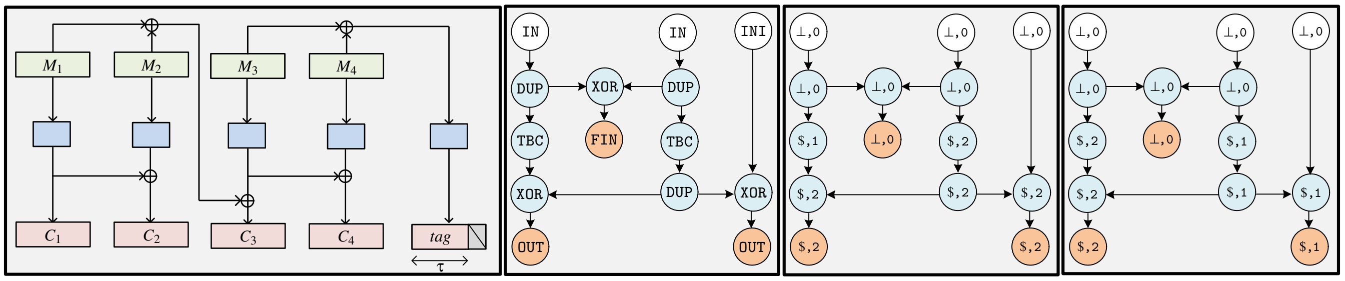

Figure 3.8: Left: A scheme that can be accepted or (incorrectly) rejected by Priv, depending on the topological ordering of the nodes. Middle left: The corresponding Enc graph. Middle right: The $(type, ctr)$ pairs in each node of the Enc graph if the left TBC node is visited first. The graph is (incorrectly) rejected because the two OUT nodes both have $ctr = 2$. Right: The $(type, ctr)$ pairs in each node of the Enc graph if the right TBC is visited first. This time, the graph is accepted because the two OUT nodes have different ctr values.

The authenticity check for a scheme (Enc, Dec, Tag) is more complicated. We now argue informally that if a scheme passes the checks of algorithm Auth (cf. Figure 3.7), then the scheme satisfies authenticity. To see this, consider a candidate forgery (N, A, C) output by an adversary. First suppose there was no prior query $(N, A, \star)$ to the encryption oracle. Auth verifies that the Tag algorithm outputs a random tag when the tweak for the TBC node in Tag was not used previously; thus, the candidate forgery will be invalid except with probability $2^{-\tau}$. (Recall that τ is the tag length.) Next, consider the case that there was a prior encryption query (N, A, M), and let C' be the corresponding ciphertext. Then $C \neq C'$; otherwise (N, A, C) is not a valid forgery. If C and C' only differ in their tags, the candidate forgery must be invalid because the tag is uniquely determined by N, A, and the rest of the ciphertext. Otherwise, consider the first pair of blocks in which C and C' differ. Auth verifies that (i) the first half of the final state produced by Dec when run on those blocks is random, (ii) Dec has the property that if the first half of its initial state is random, then the first half of the final state it outputs is random, and (iii) Tag has the property that if the first half of its initial state is random, then the tag it outputs is random[8]. Taken together, these imply that the tag will be random, and hence the candidate forgery will be invalid except with probability $2^{-\tau}$.

To demonstrate the strength of our approach, consider a modified version of the OTR scheme [21]. The original OTR scheme (cf. Figure 3.9) is secure, which our automated tests confirm. If, however, the scheme is changed so that Σ is computed as the checksum of the *odd* blocks $M_1 \oplus M_3 \oplus \cdots$, rather than the even blocks, then it becomes insecure. And, indeed, the modified scheme does not pass our tests. Namely, on input $(0, 0, 1, 0)$ to Map we find that the required FIN node is typed 1 instead of \$.

Proofs of correctness. We now prove that schemes that pass our tests are secure. We first show that if Priv returns true when given the (unlabeled) graphs corresponding to the Enc and Tag components of some AE scheme, then that scheme satisfies privacy when instantiated with a secure tweakable blockcipher.

can neither prove secure nor find concrete attacks for; see Section 4.

[8]Although here we are considering just the first half of the final/initial state, if one switches to the second half then one will get the *same* set of synthesized schemes: if one changes the topological ordering in the graphs so that the first FIN/INI node becomes the second one, and vice versa, then the scheme remains the same.

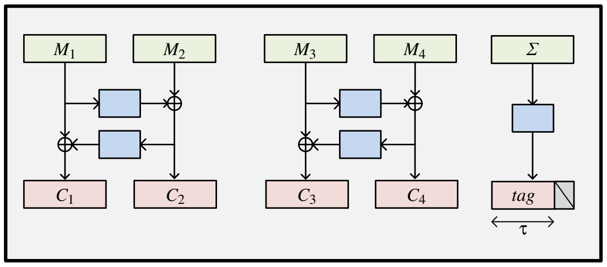

Figure 3.9: The OTR scheme, illustrated for a four-block message $M_1 \cdots M_4$. Here, Σ is the checksum of the even blocks $M_2 \oplus M_4$.

Theorem 3.3. *Let $\Pi[E] = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an AE scheme for which* $\mathsf{Priv}(G_1^-, G_2^-) = \mathsf{true}$*, where G_1^- and G_2^- are the unlabeled graphs for algorithms* Enc *and* Tag *of Π, respectively. Then for any adversary $\mathcal{A}$, there is an adversary $\mathcal{B}$ with $\mathbf{Adv}^{\mathrm{priv}}_{\Pi[E]}(\mathcal{A}) \leq \mathbf{Adv}^{\widetilde{\mathrm{prp}}}_{E}(\mathcal{B})$. Adversary $\mathcal{B}$ has the same running time as $\mathcal{A}$ and makes at most $(\mathbf{Cost}(\Pi) + 1) \cdot \sigma/2$ queries, where σ is the number of message blocks in the queries of $\mathcal{A}$.*

Proof. Adversary $\mathcal{B}$ runs $\mathcal{A}$. For each of $\mathcal{A}$'s queries (N, A, M), adversary $\mathcal{B}$ runs the encryption scheme $\Pi[E]$ on (N, A, M) with each call to E_K replaced by a query to $\mathcal{B}$'s oracle, and returns the ciphertext to $\mathcal{A}$. Finally, $\mathcal{B}$ outputs the same guess as $\mathcal{A}$. Let $\Pi[\pi]$ be the ideal variant of $\Pi[E]$, where calls to E_K are replaced by corresponding queries to π, with $\pi \leftarrow\!\!\$\ \mathrm{Perm}(\mathcal{T}, n)$. It suffices to show that $\mathbf{Adv}^{\mathrm{priv}}_{\Pi[\pi]}(\mathcal{A}) = 0$.

Consider experiments H_1–H_4 in Figure 3.10. The adversary has oracle access to the encryption scheme of $\Pi[\pi]$ in experiment H_1, and oracle access to $\$(\cdot,\cdot,\cdot)$ in experiment H_4. Experiment H_2 is identical to H_1, except that we re-sample $\pi \leftarrow\!\!\$\ \mathrm{Perm}(\mathcal{T}, n)$ each time we use Enc or Tag. Since a tweak to π is never repeated, $\Pr[H_1^{\mathcal{A}} \Rightarrow \mathsf{true}] = \Pr[H_2^{\mathcal{A}} \Rightarrow \mathsf{true}]$. In experiment H_3, instead of calling $\mathsf{Tag}^{\pi}(T, X)$ to get the tag, we sample the tag at random. Considering lines 02–03 of Priv (and the fact that $\mathsf{Priv}(G_1^-, G_2^-) = \mathsf{true}$) in conjunction with Lemma 3.1 shows that the string $V := \mathsf{Tag}^{\pi}(T, X)$ is uniform and so experiments H_2 and H_3 are identical. Finally, experiment H_4 is identical to H_3, except that instead of calling $\mathsf{Enc}^{\pi}(T, X, M_{2i-1}M_{2i})$ to get the blocks $C_{2i-1}C_{2i}$ of the ciphertext, we sample them at random. Consider-

```
proc ENCRYPT[$\pi$]($N, A, M$)          // Experiments $H_1$, $\boxed{H_2}$
$M_1 \cdots M_{2m} := M;\ X := 0^{2n};\ v := 1$     // $|M_i| = n$
for $i = 1$ to $m$ do
    $T := (N, A, v);\ \boxed{\pi \leftarrow\!\!\$\ \mathrm{Perm}(\mathcal{T}, n)}$
    $(Y, C_{2i-1}C_{2i}) := \mathsf{Enc}^{\pi}(T, X, M_{2i-1}M_{2i})$
    $v := v + \mathbf{Cost}(\Pi);\ X := Y$
$\boxed{\pi \leftarrow\!\!\$\ \mathrm{Perm}(\mathcal{T}, n);}$ $T := (N, A, 1 - v);\ V := \mathsf{Tag}^{\pi}(T, X)$
return $C_1 \cdots C_{2m} \parallel V[1, \tau]$
```

```
proc ENCRYPT[$\pi$]($N, A, M$)          // Experiments $H_3$, $\boxed{H_4}$
$M_1 \cdots M_{2m} := M;\ X := 0^{2n};\ v := 1$     // $|M_i| = n$
for $i = 1$ to $m$ do
    $T := (N, A, v);\ \pi \leftarrow\!\!\$\ \mathrm{Perm}(\mathcal{T}, n)$
    $(Y, C_{2i-1}C_{2i}) := \mathsf{Enc}^{\pi}(T, X, M_{2i-1}M_{2i})$
    $\boxed{C_{2i-1}C_{2i} \leftarrow\!\!\$\ \{0,1\}^{2n}}$
    $v := v + \mathbf{Cost}(\Pi);\ X := Y$
$T := (N, A, 1 - v);\ V \leftarrow\!\!\$\ \{0,1\}^{n}$
return $C_1 \cdots C_{2m} \parallel V[1, \tau]$
```

Figure 3.10: Experiments H_1–H_4 in the proof of Theorem 3.3. Experiments H_2 and H_4 include the corresponding boxed statements, but H_1 and H_3 do not.

ing lines 04–05 of Priv (and the fact that $\mathsf{Priv}(G_1^-, G_2^-) = \mathsf{true}$) in conjunction with Lemma 3.1 shows that the output blocks of $\mathsf{Enc}^{\pi}(T, X, M_{2i-1}M_{2i})$ are uniform and independent (and this is true even conditioned on all prior ciphertext blocks). Hence H_3 and H_4 are identical, and $\mathbf{Adv}^{\mathrm{priv}}_{\Pi[\pi]}(\mathcal{A}) = \Pr[H_1^{\mathcal{A}} \Rightarrow \mathsf{true}] - \Pr[H_4^{\mathcal{A}} \Rightarrow \mathsf{true}] = 0$. □

Next, in Theorem 3.4, we show that if Auth in Figure 3.7 returns true when given graphs corresponding to the Dec and Tag components of some AE scheme, then that scheme satisfies authenticity when instantiated with a secure tweakable blockcipher. (Examination of the proof shows that if algorithm Dec does not use E_K^{-1}, as in the case of OTR, then the term $\mathbf{Adv}^{\pm\widetilde{\mathrm{prp}}}_{E}(\mathcal{B})$ in Theorem 3.4 can be weakened to $\mathbf{Adv}^{\widetilde{\mathrm{prp}}}_{E}(\mathcal{B})$.)

THEOREM 3.4. *Let $\Pi[E] = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an AE scheme such that $\mathsf{Auth}(G_1^-, G_2^-) = \mathsf{true}$, where G_1^-, G_2^- are the unlabeled graphs for algorithms* Dec *and* Tag *of Π, respectively. Then for any adversary $\mathcal{A}$, there is an adversary $\mathcal{B}$ with $\mathbf{Adv}^{\mathrm{auth}}_{\Pi[E]}(\mathcal{A}) \le 2^{-\tau} + \ell \cdot (\mathbf{Cost}(\Pi) + 2)/2^n + \mathbf{Adv}^{\pm\widetilde{\mathrm{prp}}}_{E}(\mathcal{B})$, where ℓ is the number of blocks in the forgery output by $\mathcal{A}$. Adversary $\mathcal{B}$ has the same running time as $\mathcal{A}$ and makes at most $(\mathbf{Cost}(\Pi)+1) \cdot \sigma/2$ queries, where σ is the total number of message blocks in the queries of $\mathcal{A}$.*

PROOF. Adversary $\mathcal{B}$ runs $\mathcal{A}$. For each of $\mathcal{A}$'s encryption queries, $\mathcal{B}$ runs the encryption scheme of $\Pi[E]$ but with each call to E_K replaced by a query to $\mathcal{B}$'s first oracle, and returns the ciphertext to $\mathcal{A}$. When $\mathcal{A}$ outputs a forgery (N, A, C), adversary $\mathcal{B}$ runs the decryption scheme of $\Pi[E]$ on (N, A, C), but with each call to E_K/E_K^{-1} replaced by a query to $\mathcal{B}$'s oracles. Adversary $\mathcal{B}$ returns 1 if $\mathcal{A}$ output a valid forgery, and returns 0 otherwise. Let $\Pi[\pi]$ be the ideal variant of $\Pi[E]$, where calls to E_K/E_K^{-1} are replaced by corresponding queries to π/π^{-1}, with $\pi \leftarrow\!\!\$\ \mathrm{Perm}(\mathcal{T}, n)$. It suffices to show that $\mathbf{Adv}^{\mathrm{auth}}_{\Pi[\pi]}(\mathcal{A}) \le 2^{-\tau} + \ell(\mathbf{Cost}(\Pi) + 2)/2^n$.

Consider experiments H_1–H_3 in Figure 3.11. In H_1, the adversary has oracle access to the encryption and decryption

```
proc DECRYPT[$\pi$]($N, A, C$)          // Experiments $H_1$, $\boxed{H_2}$
if $|C| \not\equiv \tau \pmod{2n}$ then return $\perp$
$C_1 \cdots C_{2m} \parallel tag := C$  // $|C_i| = n$ and $|tag| = \tau$
$X := 0^{2n};\ v := 1$
for $i = 1$ to $m$ do
    $T := (N, A, v)$
    $(Y, M_{2i-1}M_{2i}) := \mathsf{Dec}^{\pi, \pi^{-1}}(T, X, C_{2i-1}C_{2i})$
    $v := v + \mathbf{Cost}(\Pi);\ X := Y$
$\boxed{\pi \leftarrow\!\!\$\ \mathrm{Perm}(\mathcal{T}, n)}$
$T := (N, A, 1 - v);\ V := \mathsf{Tag}^{\pi}(T, X)$
if $tag \ne V[1, \tau]$ then return $\perp$
return $M_1 \cdots M_{2m}$
```

```
proc DECRYPT[$\pi$]($N, A, C$)                    // Experiment $H_3$
if $|C| \not\equiv \tau \pmod{2n}$ then return $\perp$
$C_1 \cdots C_{2m} \parallel tag := C$  // $|C_i| = n$ and $|tag| = \tau$
$X := 0^{2n};\ v := 1$
for $i = 1$ to $m$ do
    $T := (N, A, v)$
    $(Y, M_{2i-1}M_{2i}) := \mathsf{Dec}^{\pi, \pi^{-1}}(T, X, C_{2i-1}C_{2i})$
    $v := v + \mathbf{Cost}(\Pi);\ X := Y$
$V \leftarrow\!\!\$\ \{0,1\}^{n}$
if $tag \ne V[1, \tau]$ then return $\perp$
return $M_1 \cdots M_{2m}$
```

Figure 3.11: Experiments H_1–H_3 in the proof of Theorem 3.4. Experiment H_2 includes the corresponding boxed statement, but experiment H_1 does not. Each experiment also has a procedure ENCRYPT[π], implementing the encryption algorithm of $\Pi[\pi]$, that is not shown for simplicity.

schemes of $\Pi[\pi]$. Experiment H_2 is identical to H_1, except that when running the decryption algorithm, we re-sample $\pi \leftarrow\!\!\$\ \mathrm{Perm}(\mathcal{T}, n)$ before using it in Tag. Experiment H_3 is identical to H_2, except that instead of using Tag to generate the tag, we sample the tag uniformly.

Let (N, A, C) be the forgery output by $\mathcal{A}$. Suppose there is no encryption query (N, A, M') with $|M'| = |C| - \tau$. Since decryption of the forgery query involves calling Tag with a tweak that has never been used before, we have $\Pr[\mathcal{A}\text{ forges in }H_1] = \Pr[\mathcal{A}\text{ forges in }H_2]$. Considering lines 12–13 of Auth (and the fact that $\mathsf{Auth}(G_1^-, G_2^-) = \mathsf{true}$) in conjunction with Lemma 3.1 shows that the string $V := \mathsf{Tag}^{\pi}(T, X)$ is uniform. Thus $\Pr[\mathcal{A}\text{ forges in }H_2] = \Pr[\mathcal{A}\text{ forges in }H_3]$. The probability that $\mathcal{A}$ can forge in H_3 is at most $2^{-\tau}$. Hence $\mathbf{Adv}^{\mathrm{auth}}_{\Pi[\pi]}(\mathcal{A}) \le 2^{-\tau}$ in this case.

Now, suppose that there is an encryption query (N, A, M') such that $|M'| = |C| - \tau$. (Note that there can be at most one such query, since the attacker is not allowed to re-use a nonce value in two encryption queries.) Let C' be the corresponding ciphertext output by this encryption query, and let $C = C_1 \cdots C_{2m} \parallel tag$ and $C' = C'_1 \cdots C'_{2m} \parallel tag'$. If $C_j = C'_j$ for every $j \le 2m$ then tag and tag' must be different and thus, since Tag is deterministic, the forgery is invalid. Otherwise, take the least index $r \le m$ such that $C_{2r-1}C_{2r} \ne C'_{2r-1}C'_{2r}$. Consider experiments $P_1, \ldots, P_{m-r+4}$ in Figure 3.12. In P_1, the adversary has two oracles: ENCRYPT and DECRYPT. The first implements the encryption scheme of $\Pi[\pi]$, and the second implements the decryption scheme of $\Pi[\pi]$ but returns false if the decrypted value is $\perp$ and returns true otherwise.

Let S be the subset of $\mathrm{Perm}(\mathcal{T}, n)$ such that for any $f \in S$ and query (T, X) that ENCRYPT[π](N, A, M') makes to π,

```
proc DECRYPT[$\pi$]($N, A, C$)                // Experiments $P_1$, $\boxed{P_2}$
if $|C| \not\equiv \tau \pmod{2n}$ then return $\perp$
$C_1 \cdots C_{2m} \,\|\, tag := C$  // $|C_i| = n$ and $|tag| = \tau$
$X := 0^{2n}$; $v := 1$
for $i = 1$ to $m$ do
    $T := (N, A, v)$; $\boxed{\pi \leftarrow\$ S}$
    $(Y, M_{2i-1}M_{2i}) := \mathsf{Dec}^{\pi,\pi^{-1}}(T, X, C_{2i-1}C_{2i})$
    $v := v + \mathbf{Cost}(\Pi)$; $X := Y$
$T := (N, A, 1 - v)$; $\boxed{\pi \leftarrow\$ S;}$ $V := \mathsf{Tag}^{\pi}(T, X)$
if $tag \neq V[1, \tau]$ then return false
return true
```

```
                // Experiments $P_{3+j}$, for $0 \le j \le m - r + 1$
proc DECRYPT[$\pi$]($N, A, C$)
if $|C| \not\equiv \tau \pmod{2n}$ then return $\perp$
$C_1 \cdots C_{2m} \,\|\, tag := C$  // $|C_i| = n$ and $|tag| = \tau$
$X := 0^{2n}$; $v := 1$
for $i = 1$ to $m$ do
    $T := (N, A, v)$; $\pi \leftarrow\$ S$
    $(Y, M_{2i-1}M_{2i}) := \mathsf{Dec}^{\pi,\pi^{-1}}(T, X, C_{2i-1}C_{2i})$
    if $r \le i \le r + j$ then $Y \leftarrow\$ \{0,1\}^n$
    $v := v + \mathbf{Cost}(\Pi)$; $X := Y$
$\pi \leftarrow\$ S$; $V := \mathsf{Tag}^{\pi}(T, X)$
if $j = m - r + 1$ then $V \leftarrow\$ \{0,1\}^n$
if $tag \neq V[1, \tau]$ then return false
return true
```

Figure 3.12: Experiments $P_1, \ldots, P_{4+m-r}$ in the proof of Theorem 3.4. Experiment P_2 includes the corresponding boxed statement, but P_1 does not. Each experiment also has a procedure ENCRYPT[π], implementing the encryption algorithm of $\Pi[\pi]$, that is not shown for simplicity. Here S is the set of $f \in \mathrm{Perm}(\mathcal{T}, n)$ such that for any query (T, X) that ENCRYPT[π](N, A, M') makes to π, it holds that $f(T, X) = \pi(T, X)$.

we have $f(T, X) = \pi(T, X)$. Experiment P_2 is identical to P_1, except that in procedure DECRYPT, each time we call Dec or Tag we resample $\pi \leftarrow\$ S$. Since in the forgery query we do not repeat the tweak of any encryption query other than (N, A, M'), and π and π^{-1} are called with distinct tweaks, we have $\Pr[\mathcal{A}$ forges in $P_1] = \Pr[\mathcal{A}$ forges in $P_2]$. In experiment P_3 we sample Y uniformly instead of computing $Y := \mathsf{Dec}^{\pi,\pi^{-1}}(T, X, C_{2r-1}C_{2r})$. Considering lines 14–16 of Auth (and the fact that $\mathsf{Auth}(G_1^-, G_2^-) = \mathsf{true}$) in conjunction with Lemma 3.2, we have $\Pr[\mathcal{A}$ forges in $P_2] - \Pr[\mathcal{A}$ forges in $P_3] \le \frac{2\mathbf{Cost}(\Pi)+2}{2^n}$.

For $j = 1, \ldots, m - r$, experiment P_{3+j} is identical to P_{2+j}, except that we sample Y uniformly instead of computing $Y := \mathsf{Dec}^{\pi,\pi^{-1}}(T, X, C_{2r+2j-1}C_{2r+2j})$. Considering lines 17–19 of Auth (and the fact that $\mathsf{Auth}(G_1^-, G_2^-) = \mathsf{true}$) in conjunction with Lemma 3.2, we conclude that $\Pr[\mathcal{A}$ forges in $P_{2+j}] - \Pr[\mathcal{A}$ forges in $P_{3+j}] \le \frac{2\mathbf{Cost}(\Pi)+2}{2^n}$.

Experiment P_{m-r+4} is identical to P_{m-r+3} except that we sample V uniformly when checking the validity of the forgery instead of computing $V := \mathsf{Tag}^{\pi}(T, X)$. Let X' be the state used by Tag in ENCRYPT[π](N, A, M'). If $X[1, n] \neq X'[1, n]$, which happens with probability at least $1 - 2^{-n}$, then applying Lemma 3.2 to lines 20–22 of procedure Auth, we have $\Pr[\mathcal{A}$ forges in $P_{m-r+3}] - \Pr[\mathcal{A}$ forges in $P_{m-r+4}] \le \frac{2}{2^n}$. Finally, $\Pr[\mathcal{A}$ forges in $P_{m-r+4}] \le 2^{-\tau}$. Summing up,

$$\begin{aligned}\mathbf{Adv}^{\mathrm{auth}}_{\Pi[\pi]}(\mathcal{A}) &\le 2^{-\tau} + \frac{2(m - r + 1)(\mathbf{Cost}(\Pi) + 1) + 3}{2^n} \\ &\le 2^{-\tau} + \frac{\ell(\mathbf{Cost}(\Pi) + 2)}{2^n}. \qquad \square\end{aligned}$$

To summarize, Theorems 3.3 and 3.4 show that if the graphs induced by a given scheme Π satisfy Priv and Auth as defined in Figure 3.7, then Π is a secure AE scheme.

4. IMPLEMENTATION AND RESULTS

We have implemented the Priv and Auth algorithms described in Section 3, and used them to synthesize AE schemes. The code is written in OCaml and available at `https://github.com/amaloz/ae-generator`. Our system has two modules: an *analysis module* that, given graphs corresponding to an AE scheme, verifies whether the scheme is secure, and a *synthesis module* that synthesizes AE schemes by enumerating candidate AE schemes and using the analysis module to see if they are secure. We describe these components below, where throughout this section, the term *graph* denotes an unlabeled graph.

Analyzer. The analysis module takes as input a representation (in a stack-based language) of the Dec and Tag graphs; the stack-based language makes it easy to both convert the inputs into their respective graphs as well as to synthesize schemes. We first derive a graph for the Enc algorithm given the graph for the Dec algorithm, as described below. Given graphs for the Enc, Dec, and Tag algorithms, we can then run the privacy and authenticity checks described in Figure 3.7 to check security of the scheme. Our analyzer is able to verify simplified variants of OCB [23], XCBC [12], COPA [3], OTR [21], and CCM [10], among others.

Deriving the Enc graph. We implement an algorithm Reverse that, given a Dec graph, computes a corresponding Enc graph if one exists. The basic idea is to swap the `IN` and `OUT` nodes of the input graph (recall that `IN` and `OUT` nodes in the Dec graph denote ciphertext blocks and plaintext blocks, respectively, whereas `IN` and `OUT` nodes in the Enc graph are flipped), and then selectively reverse the edges to ensure that each node has correct ingoing/outgoing degrees. Deriving the Enc graph is thus simple if there is at most one path from an `IN` node to an `OUT` node and these paths do not cross, as in the case of OCB. However, in other schemes, such as OTR, each `IN` node may have multiple paths to each `OUT` node. We handle this as described next.

On input G_1^-, let $\mathcal{G}_1$ be the *undirected* graph of G_1^-. In $\mathcal{G}_1$, rename `IN` nodes as `OUT` nodes, and `OUT` nodes as `IN` nodes; let $\mathcal{G}_2$ be the resulting graph. Reverse then assigns direction to the edges of $\mathcal{G}_2$ such that each node has correct ingoing/outgoing degrees; the resulting graph G_2^- is output. (If no assignment is possible, then the output is $\perp$.)

To implement this idea efficiently, we color each node either "red" or "blue", where red nodes denote nodes that have already been processed, and blue nodes denote unprocessed nodes. Starting from $\mathcal{G}_2$, we initially color `IN` and `INI` nodes red and all other nodes blue. We repeatedly iterate over the blue nodes until we reach a fixed point, where in each iteration we assign direction to some edges and re-color some nodes red. If a fixed point is reached before all nodes have been colored red, we return $\perp$; otherwise, we return $\mathcal{G}_2$, which represents the reversed graph. If the graph $\mathcal{G}_2$ has r nodes then we have at most r iterations with each iteration taking $O(r)$ time.

In each iteration, we process each blue node x as follows. Let $\mathsf{ord}(x) = 2$ if x is an XOR node, and let $\mathsf{ord}(x) = 1$ otherwise. If there are *exactly* $\mathsf{ord}(x)$ red neighbors of x then (1) for each such neighbor y, assign the direction $y \to x$, and (2) color x red. Note that in each step we ensure that the current node x has the correct ingoing degree if we color it red. We never assign an ingoing edge to x in any other step. Hence when there are no blue nodes, each node in the directed graph has the correct ingoing/outgoing degrees.

We prove in the full version [13] that Reverse is sound; namely, that if running Reverse on a Dec graph produces an Enc graph, then Dec is a correct decryption algorithm for Enc.

As a side note, the Reverse algorithm allows us to easily check if a scheme is *inverse-free* (i.e., the scheme only uses the forward direction of the TBC), which is important when constructing hardware realizations of AE schemes due to the potential savings in chip space, among other benefits [15, 21]. After running Reverse, we can check if the parent nodes for all the TBC nodes in the Enc and Dec graph are the same; if so, the scheme is inverse-free.

Synthesizer. We synthesize schemes as follows. Fixing a Tag graph, we enumerate all possible Dec graphs of a given size, pruning out "uninteresting" schemes such as ones with two (or more) TBC nodes chained together, and feed each pair of (Dec, Tag) graphs to our analysis module. To generate the Dec graph, we start from a graph containing just the IN and INI nodes, and add nodes and their corresponding edges until the given size bound is reached. If the resulting graph is "well-formed" (i.e., there are no "dangling" edges and no loops), we derive the corresponding Enc graph as discussed above and run the analysis module on the result. Unfortunately, this approach is prohibitively expensive as described, especially as the size bound increases. Thus, we use several optimizations to speed up the process.

Firstly, instead of synthesizing graphs with FIN and OUT nodes, we replace these with "terminal" nodes. Upon deriving a well-formed graph, we replace the "terminal" nodes with all possible permutations of FIN and OUT nodes and check security of each. Thus we no longer need to explore the search space for each FIN and OUT node; instead, we explore the search space *once* using a "terminal" node, and later replace the "terminal" node with all possible combinations of FIN and OUT nodes. Likewise, we can apply this same idea to INI and IN nodes by introducing a "start" node.

Secondly, we observe that AE schemes like OCB, COPA, and OTR do not utilize one of the INI nodes in the sense that they simply output the input value directly. Thus, we can remove two nodes from the synthesis by only synthesizing schemes containing one INI and FIN node. The drawback of this optimization is that it misses schemes such as XCBC and CCM which do in fact use both INI nodes; however, it greatly speeds up synthesis. All the results that follow use this optimization. (It would, of course, be possible to synthesize schemes without using this optimization.)

Results. Using the optimizations described above, we ran our synthesizer to find AE schemes with Dec and Enc graphs of sizes between twelve and sixteen (we found no AE schemes with size less than twelve). Note that our synthesizer does not remove duplicate schemes. In addition, there are many "equivalent" schemes in the sense that one is the same as another except with the outputs and/or inputs flipped. We thus developed a heuristic to remove duplicate and "equivalent" schemes as follows. Let $F(\cdot,\cdot,\cdot)$ be the encrypt operation of a given scheme, where the first argument is the INI input (recall we consider the simplified variant where we only use one INI node) and the other arguments are the IN inputs. Choosing arbitrary but fixed inputs X, M_1, and M_2, we compute $Y \| C_1 \| C_2 := F(X, M_1, M_2)$ and $Y' \| C_1' \| C_2' := F(X, M_2, M_1)$. We maintain a table of existing ciphertexts; if any of YC_1C_2, YC_2C_1, $Y'C_1'C_2'$, or $Y'C_2'C_1'$ exists in the table, we discard the scheme as a "duplicate"; otherwise, we add each of these to the table and continue.

#	Unique	"Optimal"	WP	SP	Time
12	13 (0)	13	7	5	47 sec
13	142 (0)	0	0	0	4.3 min
14	582 (2)	171	48 (4)	5	24.2 min
15	2826 (54)	40	18	6	2.8 hours
16	3090 (—)	66	25 (4)	1	3 hours*
Total	6653	290	98 (8)	17	

Figure 4.1: Synthesis results. The first column shows the number of instructions in the Dec graph of the given scheme; the second column the number of secure (and unique) schemes, with the number in parentheses denoting the number of schemes in which the security check fails but we cannot automatically find a concrete attack; the third column the number of (secure) schemes that are "optimal", i.e., having two TBC nodes per Dec graph; the fourth column the number of "optimal" weakly parallelizable schemes, with the number in parentheses denoting the weakly parallelizable schemes which only use the forward direction of the TBC; the fifth column the number of "optimal" *strongly* parallelizable schemes; and the final column the total synthesis time, where an asterisk indicates that we halted execution after the given time.

Figure 4.1 shows the results. The experiments were run on a commodity laptop; because of the long running time for synthesizing schemes of size sixteen, we stopped the synthesis after three hours for this size. Due to the large number of discovered schemes, we developed two algorithms to prune the result space. The first simply filters out all schemes Π such that $\mathbf{Cost}(\Pi) > c$ for some integer c. In Figure 4.1 we set $c = 2$, thus pruning out all non-"rate-1" schemes; this removes 95% of the found schemes.

Our second algorithm checks whether a scheme is *parallelizable*, an important criterion for AE schemes. Note that we can view the encryption of a message $M = M_1 \cdots M_{2m}$ as a single graph constructed from m Enc graphs $G_1, \ldots, G_m$, where the FIN nodes of G_i coincide with the INI nodes of G_{i+1}. We can then assign a "depth" to each node in this graph as follows: (1) The INI nodes in G_1 and the IN nodes in $\{G_i\}$ get a depth of 0; (2) For each node x, let t be the maximum depth of x's parent(s); if x is a TBC node then $\mathsf{depth}(x) = t + 1$; otherwise $\mathsf{depth}(x) = t$. (Intuitively, $\mathsf{depth}(x)$ represents the *latency*, in terms of the number of TBC calls, of computing the value at node x.) We can use the same idea to compute a "depth" for decryption.

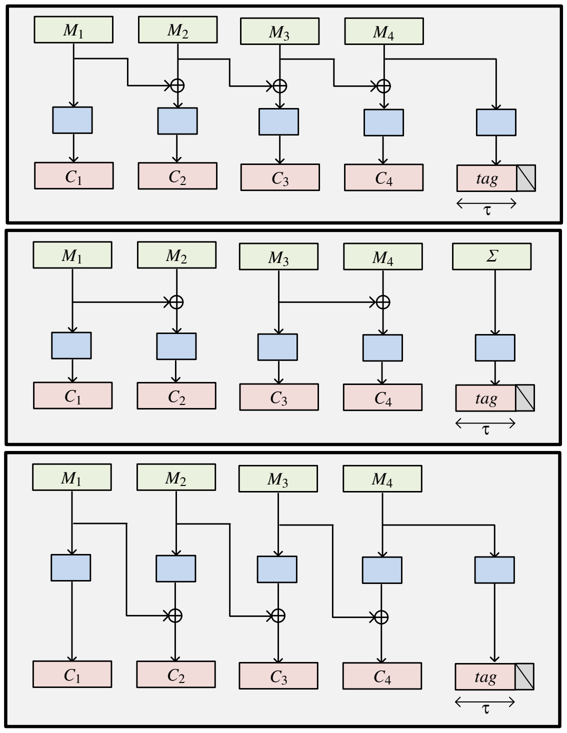

Figure 4.2: Three of our synthesized schemes of size twelve, illustrated for a four-block message $M_1, \ldots, M_4$. In the second scheme, Σ is the checksum of the even blocks, i.e., $\Sigma = M_2 \oplus M_4$.

We now define our notion of parallelizability. We call an AE scheme Π *weakly parallelizable* if for any integer m and any node x in the graph described above we have $\mathsf{depth}(x) \leq \mathbf{Cost}(\Pi)$ for both encryption and decryption. A scheme is *strongly parallelizable* if $\mathsf{depth}(x) \leq 1$. Intuitively, weakly parallelizable schemes are ones where the TBC calls can be parallelized *across* two-block chunks (but not necessarily within the processing of a two-block chunk), and strongly parallelizable schemes are ones where the TBC calls can be parallelized even *within* a two-block chunk. As an example, OTR (cf. Figure 3.9) is weakly parallelizable while OCB (cf. Figure 3.2) is strongly parallelizable.

We can check these conditions efficiently by noting first that we only need to look at the `OUT` and `FIN` nodes, since depth is strictly increasing. Now, suppose we run the analysis on graph G_i. If the depth of the `FIN` nodes is zero, then it suffices to compute $t = \max\{\mathsf{depth}(\mathtt{OUT}_1), \mathsf{depth}(\mathtt{OUT}_2)\}$ (where $\mathtt{OUT}_1$ and $\mathtt{OUT}_2$ are the two `OUT` nodes) and check whether $t \leq \mathbf{Cost}(\Pi)$ or $t \leq 1$. If the `FIN` nodes have depth greater than zero (say, c), we need to rerun the analysis, this time setting the depths of the `INI` nodes to c (rather than zero). We can then compute $t' = \max\{\mathsf{depth}(\mathtt{OUT}_1), \mathsf{depth}(\mathtt{OUT}_2)\}$. If $t' \neq t$ then this implies that depth grows with m (and thus the scheme is not parallelizable); otherwise we can check whether $t' \leq \mathbf{Cost}(\Pi)$ or $t' \leq 1$ to determine whether the scheme is parallelizable.

Looking at the results of Figure 4.1, we found thirteen secure AE schemes of size twelve, five of which are strongly parallelizable. Of these schemes, as far as we know, only OCB exists in the literature. In Figure 4.2 we show two of these newly synthesized schemes, along with one which is *not* parallelizable. (For all the schemes in the figure, encryption is strongly parallelizable; for the third scheme, however, decryption cannot be parallelized.) We implemented all three schemes and compared their performance with that of an optimized implementation of OCB by Krovetz[9] using AES-NI; see Figure 4.3. (Note that the results in Figure 4.3 are preliminary timing numbers; the purpose of these experiments is to show that our schemes are competitive with, not necessarily better than, OCB.) We find that the encryption procedure for all four schemes is comparable. However, the decryption procedure of the third synthesized scheme is noticeably slower than the others. This is because decryption for this scheme is not parallelizable; namely, to decrypt ciphertext block C_i we need plaintext block M_{i-1}.

Scheme	Enc (cycles/byte)	Dec (cycles/byte)
OCB	0.7122 ± 0.0072	0.7650 ± 0.0025
1	0.7253 ± 0.0055	0.7485 ± 0.0047
2	0.7116 ± 0.0025	0.7643 ± 0.0023
3	0.8139 ± 0.0121	2.7566 ± 0.0010

Figure 4.3: Performance results of OCB and the three synthesized schemes in Figure 4.2 (Scheme 1 denotes the top scheme, Scheme 2 the middle scheme, and Scheme 3 the bottom scheme). We report the time for encryption and decryption when processing a 4096-bit message with empty associated data, along with the 95% confidence intervals over 100 runs of each scheme. The experiments were run on a 4-core 2.90 GHz Intel Core i5-4210H CPU with TurboBoost disabled.

In addition, among the weakly parallelizable schemes, we found eight schemes which are inverse-free (we found no such schemes for strongly parallelizable schemes). The schemes of size fourteen that we found use one fewer XOR instruction than OTR, the fastest known inverse-free AE scheme we are aware of.

We also ran our attack generation algorithm over schemes of size 12–15 and found that the number of schemes where no attack could be found closely matched the number of schemes our analysis found secure, thus pointing to the fact that while our analysis is not sound, it appears to capture most secure schemes; see the full version [13] for details.

We remark that our tool currently takes a given bound S and enumerates all schemes in which decryption can be implemented using at most S instructions. In future work one could consider assigning a cost to different instructions (e.g., letting `DUP` have cost 0, and letting `TBC` have cost some fixed multiple of `XOR`) and enumerating all schemes having at most some given cost.

5. CONCLUSION

In this work, we present a methodology for automatically proving the security of a large class of authenticated encryption (AE) schemes. Using our approach, we are able to synthesize thousands of schemes, most of which have never been studied in the literature. Among these, we discovered five new schemes which are as "compact" (in terms of the number of instructions per message block), as "efficient" (in terms of the number of blockcipher calls per message block), and as parallelizable as OCB, with competitive performance.

There are several interesting avenues for future work. Further optimizing the synthesis procedure would allow us to

[9]See `http://web.cs.ucdavis.edu/~rogaway/ocb/news`.

generate more schemes. Some of these schemes may have additional properties of interest, such as misuse-resistance [11]; developing techniques for automatically checking schemes for these additional properties would be very useful. Taking a different approach, it would be interesting to see if similar techniques can be applied to more general classes of AE schemes.

Acknowledgments

We thank Samuel Neves for pointing out that on Haswell CPUs, TurboBoost must be disabled to measure the timing accurately. We also thank the anonymous reviewers of CCS 2015 for their suggestion of automatic generation of attacks on schemes that fail our tests.

Work of Jonathan Katz was done for Exelis under contract number N00173-11-C-2045 to NRL. Work of Alex J. Malozemoff was conducted with Government support awarded by DoD, Air Force Office of Scientific Research, National Defense Science and Engineering Graduate (NDSEG) Fellowship, 32 CFR 168a.

6. REFERENCES

[1] Joseph A. Akinyele, Matthew Green, and Susan Hohenberger. Using SMT solvers to automate design tasks for encryption and signature schemes. In *ACM CCS 2013*, pages 399–410, November 2013.

[2] Joseph A. Akinyele, Matthew Green, Susan Hohenberger, and Matthew W. Pagano. Machine-generated algorithms, proofs and software for the batch verification of digital signature schemes. In *ACM CCS 2012*, pages 474–487, October 2012.

[3] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Elmar Tischhauser, and Kan Yasuda. Parallelizable and authenticated online ciphers. In *Asiacrypt 2013*, pages 424–443, December 2013.

[4] Gilles Barthe, Juan Manuel Crespo, Benjamin Grégoire, César Kunz, Yassine Lakhnech, Benedikt Schmidt, and Santiago Zanella Béguelin. Fully automated analysis of padding-based encryption in the computational model. In *ACM CCS 2013*, pages 1247–1260, November 2013.

[5] Gilles Barthe, Edvard Fagerholm, Dario Fiore, John C. Mitchell, Andre Scedrov, and Benedikt Schmidt. Automated analysis of cryptographic assumptions in generic group models. In *Crypto 2014*, pages 95–112, August 2014.

[6] Gilles Barthe, Edvard Fagerholm, Dario Fiore, Andre Scedrov, Benedikt Schmidt, and Mehdi Tibouchi. Strongly-optimal structure preserving signatures from type II pairings: Synthesis and lower bounds. In *PKC 2015*, pages 355–376, March / April 2015.

[7] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *Journal of Cryptology*, 21(4):469–491, October 2008.

[8] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *Eurocrypt 2006*, pages 409–426, May / June 2006.

[9] Dan Bernstein. Cryptographic competitions: CAESAR call for submissions, final (2014.01.27). `http://competitions.cr.yp.to/caesar-call.html`.

[10] Morris Dworkin. Recommendations for block cipher modes of operation: The CCM mode for authentication and confidentiality. NIST Special Publication 800-38C, July 2007.

[11] Ewan Fleischmann, Christian Forler, and Stefan Lucks. McOE: A family of almost foolproof on-line authenticated encryption schemes. In *FSE 2012*, pages 196–215, March 2012.

[12] Virgil D. Gligor and Pompiliu Donescu. Fast encryption and authentication: XCBC encryption and XECB authentication modes. In *FSE 2001*, pages 92–108, April 2002.

[13] Viet Tung Hoang, Jonathan Katz, and Alex J. Malozemoff. Automated analysis and synthesis of authenticated encryption schemes. Cryptology ePrint Archive, Report 2015/624, 2015. `https://eprint.iacr.org/2015/624`.

[14] Tetsu Iwata, Keisuke Ohashi, and Kazuhiko Minematsu. Breaking and repairing GCM security proofs. In *Crypto 2012*, pages 31–49, August 2012.

[15] Tetsu Iwata and Kan Yasuda. BTM: A single-key, inverse-cipher-free mode for deterministic authenticated encryption. In *SAC 2009*, pages 313–330, August 2009.

[16] Jonathan Katz and Moti Yung. Unforgeable encryption and chosen ciphertext secure modes of operation. In *FSE 2000*, pages 284–299, April 2001.

[17] Ted Krovetz and Phillip Rogaway. The software performance of authenticated-encryption modes. In *FSE 2011*, pages 306–327, February 2011.

[18] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable block ciphers. In *Crypto 2002*, pages 31–46, August 2002.

[19] Alex J. Malozemoff, Jonathan Katz, and Matthew D. Green. Automated analysis and synthesis of block-cipher modes of operation. In *IEEE CSF 2014*, pages 140–152, July 2014.

[20] David A. McGrew and John Viega. The security and performance of the Galois/counter mode (GCM) of operation. In *Indocrypt 2004*, pages 343–355, December 2004.

[21] Kazuhiko Minematsu. Parallelizable rate-1 authenticated encryption from pseudorandom functions. In *Eurocrypt 2014*, pages 275–292, May 2014.

[22] Phillip Rogaway. Authenticated-encryption with associated-data. In *ACM CCS 2002*, pages 98–107, November 2002.

[23] Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In *Asiacrypt 2004*, pages 16–31, December 2004.

[24] Phillip Rogaway. Nonce-based symmetric encryption. In *FSE 2004*, pages 348–359, February 2004.

[25] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: A block-cipher mode of operation for efficient authenticated encryption. In *ACM CCS 2001*, pages 196–205, November 2001.

[26] Ashish Tiwari, Adrià Gascón, and Bruno Dutertre. Program synthesis using dual interpretation. In *CADE 2015*, August 2015.