1 Message Authentication

We start with some simple questions from real life:

- How do you know an email is from who it claims to be from? (Do you?)
- How does a soldier know his orders come from his commander?
- How does a bartender know you're 21?

All of these are questions of *message authentication*: determining whether the contents of a message (an email, a soldier's orders, an identification card) came from their supposed source, without being modified (or generated outright) by a malicious interloper.

A first natural question is: does encryption solve the problem of message authentication?

Here is a proposal for the setting where Alice and Bob share a secret key: to authenticate a message, Alice simply encrypts the message m under a secure shared-key scheme. For example, using the one-time pad, Alice sends:

$$c \leftarrow \underbrace{m}_{\text{message}} \oplus \underbrace{k}_{\text{key}}.$$

Bob decrypts the message and decides whether it "looks good" (e.g., whether it is properly formatted and readable).

A moment's thought reveals the following property: *An adversary can flip any desired bit(s) of the message, without even knowing what the message is!* If the adversary knows anything about the format of the message, or its likely contents in certain places (which is the case in most applications), then it can change the message without causing any suspicion on the part of Bob.

Hence, although the one-time pad is *perfectly secret*, it is in a sense also *perfectly inauthentic*. The moral of this exercise is: Encryption is not *sufficient* for authentication. Later we will see that neither is it *necessary*. We need an entirely new model and security definition.

1.1 Model

We want the sender to be able to attach a "tag" or "signature" to every message, which can be checked for validity by the receiver. In the shared-key setting, a *message authentication code* MAC for message space $\mathcal{M}$ is made up of algorithms Gen, Tag, Ver:

- Gen outputs a key k.
- $\mathsf{Tag}_k(m) := \mathsf{Tag}(k, m)$ outputs a tag $t \in \mathcal{T}$ (the "tag space").
- $\mathsf{Ver}_k(m, t) := \mathsf{Ver}(k, m, t)$ either accepts or rejects (i.e., outputs 1 or 0, respectively).

For completeness, we require that for any $m \in \mathcal{M}$, for $k \leftarrow \mathsf{Gen}$ and $t \leftarrow \mathsf{Tag}_k(m)$, we have $\mathsf{Ver}_k(m, t) = 1$.

1.2 Information-Theoretic Treatment

The following definition is the analogue of perfect secrecy for authenticity, which captures the intuition that given a valid message-tag pair, no forger should be able to find a convincing tag for a *different* message.

Definition 1.1 (Perfect unforgeability). A MAC scheme is *perfectly unforgeable* if, for all (possibly unbounded) $\mathcal{F}$ and all $m \in \mathcal{M}$,

$$\mathbf{Adv}_{\mathrm{MAC}}(\mathcal{F}) := \Pr_{k \leftarrow \mathsf{Gen}, t \leftarrow \mathsf{Tag}(m)}[\mathcal{F}(m,t) \text{ succeeds}] \leq \frac{1}{|\mathcal{T}|},$$

where "succeeds" means that $\mathcal{F}$ outputs some $(m', t') \in \mathcal{M} \times \mathcal{T}$ such that $\mathsf{Ver}_k(m', t') = 1$ and $m \neq m'$.

If the above holds for the relaxed success condition requiring only that $(m', t') \neq (m, t)$ (i.e., $\mathcal{F}$ even succeeds if it outputs a *different* tag for the *same* message), then the scheme is said to be *strongly* (and perfectly) unforgeable.

Some remarks and observations about the definition:

- For any message m', the forger $\mathcal{F}$ can always just guess a tag t' uniformly at random. A valid tag (that makes $\mathsf{Ver}_k(m', t')$ accept) must exist by correctness, so $1/|\mathcal{T}|$ is the strongest bound on the forger's advantage that we can hope for.
- Unlike with encryption, the Tag_k algorithm can be *deterministic* (and all the examples we see today will be).
- If Ver_k accepts a *unique* tag for any given message, then the MAC is perfectly unforgeable if and only if it is *strongly* perfectly unforgeable.

1.2.1 Construction

We'll construct a (strongly) perfectly unforgeable MAC using a special kind of hash function family.

Definition 1.2. A family of functions $\mathcal{H} = \{h_k \colon \mathcal{M} \to \mathcal{T}\}$ is *pairwise independent* if for any *distinct* $m, m' \in \mathcal{M}$, the random variable $(h(m), h(m'))$ is uniform over $\mathcal{T}^2$, for a uniformly random $h \leftarrow \mathcal{H}$.

Example 1.3. Let p be a prime. Define $h_{a,b}(x) = ax + b \bmod p$ for $(a, b) \in \mathbb{Z}_p^2$. We let $\mathcal{M} = \mathcal{T} = \mathbb{Z}_p$. We check that the family $\{h_{(a,b)}\}$ is pairwise independent: for any *distinct* $m, m' \in \mathbb{Z}_p$, and any $t, t' \in \mathbb{Z}_p$,

$$\Pr_{(a,b) \leftarrow \mathbb{Z}_p^2}\left[h_{a,b}(m) = t \wedge h_{a,b}(m') = t'\right] = \Pr_{(a,b) \leftarrow \mathbb{Z}_p^2}\left[am + b = t \wedge am' + b = t'\right] =$$

$$\Pr_{(a,b) \leftarrow \mathbb{Z}_p^2}\left[\begin{pmatrix} m & 1 \\ m' & 1 \end{pmatrix}\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} t \\ t' \end{pmatrix}\right] = \Pr_{(a,b) \leftarrow \mathbb{Z}_p^2}\left[\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} m & 1 \\ m' & 1 \end{pmatrix}^{-1}\begin{pmatrix} t \\ t' \end{pmatrix}\right] = \frac{1}{p^2}.$$

Here we note that the matrix $\left(\begin{smallmatrix} m & 1 \\ m' & 1 \end{smallmatrix}\right)$ is invertible modulo p by our assumption that $m \neq m'$. Our family is therefore pairwise independent, as claimed.

From Definition 1.2, it is immediate to see that for any distinct $m, m' \in \mathcal{M}$ and any $t, t' \in \mathcal{T}$,

$$\Pr_{h \leftarrow \mathcal{H}}\left[h(m') = t' \mid h(m) = t\right] = \frac{1}{|\mathcal{T}|}.$$

Our MAC construction is as follows: let $\mathcal{H} = \{h_k \colon \mathcal{M} \to \mathcal{T}\}$ be a pairwise independent hash family.

- Gen outputs $h_k \leftarrow \mathcal{H}$.

- $\mathsf{Tag}_k(m)$ outputs $h_k(m) \in \mathcal{T}$.
- $\mathsf{Ver}_k(m, t)$ accepts if $t = h_k(m)$, and rejects otherwise.

Theorem 1.4. *The MAC described above is strongly, perfectly unforgeable.*

Proof. Since Ver_k accepts at most one tag for a given message, it suffices to prove that MAC is perfectly unforgeable. Now for any $m \in \mathcal{M}$, we see that

$$\mathbf{Adv}_{\mathrm{MAC}}(\mathcal{F}) := \Pr_{k \leftarrow \mathsf{Gen}, t \leftarrow \mathsf{Tag}(m)}[\mathcal{F}(m,t) \text{ succeeds}] = \Pr_{k \leftarrow \mathsf{Gen}}[\mathcal{F}(m, h_k(m)) = (m', h_k(m')) \wedge m' \neq m].$$

Now breaking up the probability space over the forger's choice of m' and t', the above expression becomes

$$\begin{aligned}
&\sum_{\substack{(m',t') \in \mathcal{M} \times \mathcal{T} \\ m' \neq m}} \Pr_{k \leftarrow \mathsf{Gen}}[\mathcal{F}(m,t) = (m',t') \wedge h_k(m') = t' \mid h_k(m) = t] \\
&\qquad = \sum_{\substack{(m',t') \in \mathcal{M} \times \mathcal{T} \\ m' \neq m}} \Pr[\mathcal{F}(m,t) = (m',t')] \cdot \Pr_{k \leftarrow \mathsf{Gen}}[h_k(m') = t' \mid h_k(m) = t] \\
&\qquad = \sum_{\substack{(m',t') \in \mathcal{M} \times \mathcal{T} \\ m' \neq m}} \Pr[\mathcal{F}(m,t) = (m',t')] \cdot \frac{1}{|\mathcal{T}|},
\end{aligned}$$

where the first equality is because $\mathcal{F}$'s random coins are independent of the choice of $k \leftarrow \mathsf{Gen}$, and the second inequality is because $\mathcal{H}$ is pairwise independent. We can immediately upper bound the final expression above by $1/|\mathcal{T}|$, as desired. □

Question 1. Although perfectly unforgeable, the above scheme is only one-time. Suppose $\mathcal{H}$ is the concrete hash family described in Example 1.3 and show how an attacker with access to two message-tag pairs generated from the same key can forge a new message-tag pair.

Question 2. Just as a pairwise-independent hash function allowed us to construct a one-time perfectly unforgeable MAC, a d-wise independent hash family can be used to form a $(d-1)$-time perfectly unforgeable MAC in precisely the same way (the security proof is essentially the same). Using Example 1.3 as a template, describe how one could construct such a family.

1.2.2 Critiques

As noted in Question 1, the scheme is only one-time! This problem is a deficiency in our *definition* of unforgeability, because it does not comport with the fact that we would like to use a MAC scheme to tag multiple messages. However, it can be shown that perfect (information-theoretic) unforgeability is impossible to achieve, if the forger gets to see enough message-tag pairs relative to the (fixed) secret key size. So to have any hope of achieving security, we need to consider a computational definition instead.

1.3 Computational Treatment

We want to be conservative and capture the fact that the adversary (forger) can choose to see tags on *arbitrary* messages of its choice, but still cannot forge a tag for another message. As usual, we do so by giving the adversary oracle access to the scheme.

Definition 1.5 (Unforgeability under Chosen-Message Attack). We say that MAC is UF-CMA if for all nuppt $\mathcal{F}$,

$$\mathbf{Adv}_{\text{MAC}}(\mathcal{F}) := \Pr_{k \leftarrow \mathsf{Gen}}\left[\mathcal{F}^{\mathsf{Tag}_k} \text{ succeeds}\right] \leq \text{negl}(n),$$

where here "succeeds" means that $\mathcal{F}$ outputs (m', t') such that

1. $\mathsf{Ver}_k(m', t') = 1$, and
2. (a) *Standard unforgeability:* m' was not a query to the Tag_k oracle, or
   (b) *Strong unforgeability:* (m', t') was not a query / response pair from the Tag_k oracle.

Construction for fixed-length messages. Here we use a PRF in place of pairwise independent hash family. Intuitively, this should work since PRFs "appear" uniform and independent on any polynomial number of (adaptive) queries. Letting $\mathcal{M} = \mathcal{T} = \{0,1\}^n$, and $\{f_k \colon \mathcal{M} \to \mathcal{T}\}$ be a PRF family, we construct the $\text{MAC} = (\mathsf{Gen}, \mathsf{Tag}, \mathsf{Ver})$ identically to the previous construction.

Theorem 1.6. *The above MAC is strongly unforgeable under chosen message attack.*

Proof. We shall show that attacking MAC is at least as hard as breaking the PRF family (i.e., distinguishing a function chosen at random from the family from a truly uniform, random function). Let $\mathcal{F}$ be a candidate nuppt forger against MAC. We use $\mathcal{F}$ to construct an (oracle) distinguisher $\mathcal{D}$ for the PRF game as follows. Notice that $\mathcal{D}$ needs to "simulate" the chosen-message attack for $\mathcal{F}$, by providing a Tag_k oracle; it will do so using its own oracle.

$\mathcal{D}$ is given oracle access to a function g, where $g \leftarrow \{f_k\}$ or $g \leftarrow U(\{0,1\}^n \to \{0,1\}^n)$, as in the PRF definition. $\mathcal{D}^g$ runs $\mathcal{F}$, and whenever $\mathcal{F}$ queries a message m to be tagged, $\mathcal{D}$ queries $t = g(m)$ and returns t to $\mathcal{F}$, also storing m in an internal list of queries it maintains. Finally, $\mathcal{F}$ outputs a candidate forgery (m', t'). If m' is different from all of the queries so far and $t' = g(m')$, then $\mathcal{D}$ returns 1, otherwise it returns 0.

We observe that D is clearly nuppt. We also remark that $\mathcal{D}$ only outputs 1 when $\mathcal{F}$ outputs a tag for a message m that was not previously queried. Now we want to know the advantage of $\mathcal{D}$ against $\{f_k\}$:

$$\mathbf{Adv}_{\mathsf{PRF}}(\mathcal{D}) = \left| \Pr_{g \leftarrow \{f_k\}}[D^g = 1] - \Pr_{g \leftarrow U(\{0,1\}^n \to \{0,1\}^n)}[D^g = 1] \right|.$$

1. When $g \leftarrow \{f_k\}$, we see that D^g emulates the chosen-message attack to $\mathcal{F}$, and accepts exactly when $\mathcal{F}$ succeeds according to the definition. Hence $\Pr[D^g = 1] = \mathbf{Adv}_{\text{MAC}}(\mathcal{F})$.
2. When $g \leftarrow U(\{0,1\}^n \to \{0,1\}^n)$, we claim that $\Pr[D^g = 1] \leq 2^{-n}$. We note that when $\mathcal{F}$ returns a message m' different from its queries $m_1, \ldots, m_q$, the value $g(m)$ is still uniform on $\{0,1\}^n$ conditioned on the values of $g(m_1), \ldots, g(m_k)$, because g is a truly random function. Therefore, $\mathcal{F}$ can succeed at guessing $g(m')$ with probability at most 2^{-n}, as claimed.

We conclude that $\mathcal{D}$ has advantage at least $\mathbf{Adv}_{\text{MAC}}(\mathcal{F}) - 2^{-n}$ against $\{f_k\}$. Since $\{f_k\}$ is a PRF family by assumption, we must have that $\mathbf{Adv}_{\text{MAC}}(\mathcal{F}) - 2^{-n} \in \text{negl}(n) \Rightarrow \mathbf{Adv}_{\text{MAC}}(\mathcal{F}) \in \text{negl}(n)$ as needed. □

Question 3. Unfortunately, the above scheme only works for fixed-length n-bit messages where n is the security parameter. The following is an attempt to use such a MAC and construct a new MAC (MAC') that works for arbitrary length messages. Show that it is not UF-CMA secure.

- $\mathsf{Gen}' := \mathsf{Gen}$
- Tag' takes an arbitrary-length message m (whose length we denote N), pads its length to be a multiple of n, divides m into n-bit blocks $(m_1, \ldots, m_{N/n})$ and outputs the tag $t := \mathsf{Tag}(m_1)\|\mathsf{Tag}(m_2)\| \ldots \|\mathsf{Tag}(m_{N/n})$.
- Ver' is defined in the natural way.

2 Authenticated Encryption

Often Alice and Bob would want both secrecy and authenticity together in one package; this is more than just the sum of the two properties.

Our model is as follows: an authenticated encryption scheme AE is made up of Gen, Enc, Dec as usual. But Dec outputs an element of $\mathcal{M} \cup \{\perp\}$, where $\perp$ is a distinguished symbol indicating "inauthentic ciphertext," and any other output message implicitly means that the ciphertext was judged to be authentic.

Definition 2.1. AE is a secure authenticated encryption scheme if it is:

1. IND-CPA-secure as an encryption scheme, and
2. Strongly unforgeable as a MAC. That is, for all nuppt $\mathcal{F}$,

$$\Pr_{k \leftarrow \mathsf{Gen}}[\mathcal{F}^{\mathsf{Enc}_k(\cdot)} \text{ forges}] \leq \text{negl}(n),$$

where "forges" means that $\mathcal{F}$ outputs some c' where $\mathsf{Dec}_k(c') \neq \perp$, and c' is different from every response of the Enc_k oracle. (That is, the only way to obtain a valid ciphertext is to get it from the encryption oracle.)

Candidate constructions. Here we give some attempts to combine an IND-CPA secure SKC with a strongly unforgeable MAC to construct AE. In all cases, we make sure to use *independent* keys for both SKC and MAC, which will be important in any potential security proof. Take $m \in \mathcal{M}$, and define AE.Gen to choose $k_a \leftarrow$ MAC.Gen and $k_e \leftarrow$ SKC.Gen, and output key (k_a, k_e). Now consider the following encryption algorithms $\text{AE.}\mathsf{Enc}_{(k_a,k_e)}(m)$:

1. "Encrypt and tag:" output $(c \leftarrow \text{SKC.}\mathsf{Enc}_{k_e}(m), t \leftarrow \text{MAC.}\mathsf{Tag}_{k_a}(m))$. This scheme has a problem: since Tag need not have any secrecy properties, its output might leak some of the plaintext m. (For example, it is easy to construct a "pathological" secure Tag that includes the first few bits of m in its output tag.) Therefore, this scheme will not necessarily be IND-CPA secure.
2. "Tag then encrypt:" output $c \leftarrow \text{SKC.}\mathsf{Enc}_{k_e}(m\|\text{MAC.}\mathsf{Tag}_{k_a}(m))$. How should Dec operate? Can you prove the scheme secure?
3. "Encrypt then tag:" output $(c \leftarrow \text{SKC.}\mathsf{Enc}_{k_e}(m), t \leftarrow \text{MAC.}\mathsf{Tag}_{k_a}(c))$. How should Dec operate? Is the scheme secure?

Answers

Question 1. Although perfectly unforgeable, the above scheme is only one-time. Suppose $\mathcal{H}$ is the concrete hash family described in Example 1.3 and show how an attacker with access to two message-tag pairs generated from the same key can forge a new message-tag pair.

Answer. Let (m_1, t_1) and (m_2, t_2) be our two pairs. As it turns out, these two pairs are enough to recover the secret key (a, b), and from there we can forge any message we want. By the construction of our scheme we have that $t_i = h_{a,b}(m_i) = am_i + b \pmod{p}$ for $i = 1, 2$. Here we have a system of two equations with two unknowns and as such we can solve with basic linear algebra.

$$\begin{aligned}
\begin{pmatrix} m_1 & 1 \\ m_2 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} &= \begin{pmatrix} t_1 \\ t_2 \end{pmatrix} \pmod{p} \\
\begin{pmatrix} a \\ b \end{pmatrix} &= (m_1 - m_2)^{-1} \begin{pmatrix} 1 & -1 \\ -m_2 & m_1 \end{pmatrix} \begin{pmatrix} t_1 \\ t_2 \end{pmatrix} \pmod{p} \\
a &= (m_1 - m_2)^{-1}(t_1 - t_2) \pmod{p} \\
b &= (m_1 - m_2)^{-1}(m_1 t_2 - t_1 m_2) \pmod{p}
\end{aligned}$$

Question 2. Just as a pairwise-independent hash function allowed us to construct a one-time perfectly unforgeable MAC, a d-wise independent hash family can be used to form a $(d-1)$-time perfectly unforgeable MAC in precisely the same way (the security proof is essentially the same). Using Example 1.3 as a template, describe how one could construct such a family.

Answer. The key point to recognize is that the pairwise-independence of $h_{a,b} = ax + b \pmod{p}$ comes from the fact that a linear function has two degrees of freedom. Supplying one pair (m, t) such that $h_{a,b}(m) = t$ fixes a relationship between a and b but still leaves p possible combinations of what a and b could be. To go from pairwise independence to d-wise independence, we simply increase the degrees of freedom by using polynomials of degree $d - 1$. Concretely, we define our family $\mathcal{H} = \{h_k\}$ such that

$$h_{c_0,\ldots,c_{d-1}}(x) = c_0 + c_1 x + \cdots + c_{d-1}x^{d-1} = \sum_{i=0}^{d-1} c_i x^i.$$

Question 3. Unfortunately, the above scheme only works for fixed-length n-bit messages where n is the security parameter. The following is an attempt to use such a MAC and construct a new MAC (MAC′) that works for arbitrary length messages. Show that it is not UF-CMA secure.

- $\mathsf{Gen}' := \mathsf{Gen}$
- Tag' takes an arbitrary-length message m (whose length we denote N), pads its length to be a multiple of n, divides m into n-bit blocks $(m_1, \ldots, m_{N/n})$ and outputs the tag $t := \mathsf{Tag}(m_1) \| \mathsf{Tag}(m_2) \| \ldots \| \mathsf{Tag}(m_{N/n})$.
- Ver' is defined in the natural way.

Answer. This new MAC is subject to several different attacks. One attack would be for an attacker to simply permute the blocks of a particular message. That is, they could query $m_1 \| m_2$, receiving the tag $t_1 \| t_2$ and then return the new pair $(m_2 \| m_1, t_2 \| t_1)$.

Another options would be a length-extension or reduction attack whereby the attacker queries $m_1 \| m_2$, receives the tag $t_1 \| t_2$, and either shortens the message – returning the pair (m_1, t_1) – or lengthens it – returning the pair $(m_1 \| m_1 \| m_2, t_1 \| t_1 \| t_2)$.

One final attack would be to combine the blocks from two messages. That is, the attacker could query $m_1 \| m_2$, receive the tag $t_1 \| t_2$, query $m'_1 \| m'_2$, recieve the tag $t'_1 \| t'_2$ and then return the pair $(m_1 \| m'_1, t_1 \| t'_1)$