1 Digital signatures

A digital signature scheme is the public-key analog of a MAC. In a digital signature, Alice begins by generating a public key-secret key pair and shares her public key with everyone, Eve included. Now suppose Bob receives a message M from alice together with a "signature" T. Using only Alice's public key, Bob can then verify that the message M was sent by Alice and that it was Alice who sent this message.

There is one important difference between a MAC and a digital signature: digital signatures are not deniable. Suppose Bob claims to Charlie that he received a message M with a signature T from Alice, but Alice denies she has sent this message. In a digital signature scheme, Charlie can check for himself whether Bob is telling the truth: If the pair (M, T) verifies under Alice's public key, then Bob is correct. If the authenticity of M is established using a MAC, this is impossible because M is only guaranteed to be unforgeable under the secret key K shared by Alice and Bob. For Charlie to verify that Alice sent M Bob needs to give him this key K, but there is no guarantee that Bob won't be cheating by providing Charlie with some other key K' that has no relation to the actual key used by Alice and Bob.

Thus a digital signature is much like a physical signature that Alice puts on paper when she signs a contract with Bob: Not only does this tell Bob that Alice is the person bound to the contract, but Bob can also present the contract in court to prove this if necessary.

Let us first give a definition of digital signatures.

Definition 1. A digital signature scheme for messages of length m is a triple of algorithms $(Gen, Sign, Ver)$ where

- Gen is a randomized algorithm that takes no input and produces a pair of keys (SK, PK).
- $Sign$ is an algorithm that takes a secret key SK and a message M and outputs a signature $Sign(SK, M)$.
- Ver is an algorithm that takes a public key PK, a message M and a signature T and outputs 1 (for `accept`) or 0 (for `error`) such that for every message M,

$$\Pr[Ver(PK, M, Sign(SK, M)) = 1].$$

The security requirement is similar to the one for MACs: Even if Bob can obtain signatures of messages of his choice by querying an oracle, he cannot produce a forgery.

Definition 2. A digital signature scheme $(Gen, Sign, Ver)$ is (s, ε) unforgeable if for every oracle circuit $A^?$ of size at most s,

$$\Pr[A^{Sign(SK,\cdot)}(PK) \text{ produces a forgery}] \leq \varepsilon$$

where a forgery is a pair (M, T) such that $Ver(PK, M, T) = 1$.

Implementing digital signatures is somewhat more difficult than the other cryptographic functionalities we have seen so far, so we will do it in stages. To begin with, we will allow the adversary $A^?$ to make only one query to the signing oracle. We will then extend this scheme to handle an arbitrary number of queries.

2 One-time signatures

We will say a digital signature is (s, ε) *one-time secure* if it satisfies the definition of (s, ε) unforgeability, but under the additional assumption that $A^?$ makes at most one query from its oracle. (Technically we require that $A^?$ contains only one oracle gate.)

This is an extremely weak notion of security so it is important to keep in mind that one-time signatures are not our final goal. They are a "temporary" object towards getting a scheme with a much stronger notion of security.

To illustrate the idea, we'll start with something even simpler: One-time signatures for a *one-bit message*, i.e. the message can take only two values $M = 0$ or $M = 1$. Given a public key PK, the adversary can ask the oracle to see either the signature of 0 or the signature of 1. If it asks for the signature of 0 then it ought to forge the signature of 1, and vice versa.

Construction of a one-time bit signature Let $f\colon \{0,1\}^k \to \{0,1\}^k$ be a one-way function. We now describe the algorithms Gen, $Sign$, and Ver.

The algorithm Gen chooses x_0 and x_1 independently at random from $\{0,1\}^k$ and let $y_0 = f(x_0)$ and $y_1 = f(x_1)$. It outputs the key pair $SK = (x_0, x_1)$ and $PK = (y_0, y_1)$.

The algorithms $Sign$ and Ver are given by

$$Sign((x_0, x_1), M) = x_M) \qquad \text{and} \qquad Ver((y_0, y_1), M, z) = \begin{cases} 1, & \text{if } f(z) = y_M \\ 0, & \text{otherwise.} \end{cases}$$

It is easy to see that $(Gen, Sign, Ver)$ is a bit signature scheme: $f(x_M) = y_M$ for both $M = 0$ and $M = 1$.

What about security? Suppose that $A^?$ queries the signing oracle on the message 0 and the oracle returns x_0. After this, the adversary has seen x_0 as well as the public key $(y_0, y_1) = (f(x_0), f(x_1))$. Can he use this information to come up with the forgery $(1, x_1)$? Intuitively, the "information" $(x_0, f(x_0))$ should be useless to him because x_1 is independent of x_0, so knowing anything about x_0 should not help him invert $f(x_1)$. To turn this intuition into a proof we have to be careful because the choice of $A^?$'s oracle query – 0 or 1 – may depend on the public key $(f(x_0), f(x_1))$.

Claim 3. *If f is an (s, ε)-one way function, then $(Gen, Sign, Ver)$ is a $(\Omega(s), 2\varepsilon)$ one-time secure bit signature scheme.*

Proof. If $(Gen, Sign, Ver)$ is not $(\Omega(s), 2\varepsilon)$ one-time secure, then there exists an oracle circuit $A'^?$ of size at most s' that makes at most one query and such that

$$\Pr[A'^{Sign(SK,\cdot)}(PK) \text{ produces a forgery}] \geq \varepsilon'.$$

Now consider the following circuit A for inverting f:

A: On input $y \in \{0,1\}^k$,
Choose $b \sim \{0,1\}$ at random. Set $x_b = \texttt{error}, y_b = y$, $x_{\bar{b}} \sim \{0,1\}^k$ and $y_{\bar{b}} = f(x_{\bar{b}})$.
Simulate $A'^?(y_0, y_1)$. When $A'^?$ makes a query M answer with x_M.
Output the signature part of $A'^?(y_0, y_1)$.

When x is chosen at random from $\{0,1\}^k$ and $y = f(x)$, $(x_0, x_1), (y_0, y_1)$ is identically distributed to a public key-secret key pair. Since M is independent of b, we have that

$$\Pr[A'^{Sign(x_0,x_1,\cdot)}(y_0, y_1) \text{ produces a forgery and } M \neq b] \\ = \Pr[A'^{Sign(x_0,x_1,\cdot)}(y_0, y_1) \text{ produces a forgery}] \Pr[M \neq b] \geq \varepsilon'/2.$$

It remains to check that whenever $A'^{Sign(x_0,x_1,\cdot)}(y_0, y_1)$ produces a forgery and $M \neq b$, it must be that $A(y)$ is an inverse for y under f, that is $f(A(y)) = y$. Since $M \neq b$, the forgery must be of the form (b, x) with $f(x) = y_b$. But $M \neq b$ also tells us that $y_b = y$, so A outputs x such that $f(x) = y$. □

Although this proof is short it contains an important idea: by choosing M at random independently of everything else, we were able to obtain a lower bound for the probability of the event that $A'^{Sign(x_0,x_1,\cdot)}(y_0, y_1)$ produces a forgery and $M \neq b$, which allows us to invert f. If we fixed M to say 0, we could have been unlucky: Perhaps A' always queries the oracle for a signature of 0, in which case $x_M = \texttt{error}$ and the answer that A provides does not look like one coming from a signing oracle.

Signing longer messages We now show how to turn the bit signature into a one-time digital signature for longer messages. The idea is to sign each bit separately. Formally, given a bit signature scheme $(Gen, Sign, Ver)$, we define the following scheme $(Gen', Sign', Ver')$ for messages of length m:

- Key generation: Gen' runs Gen independently m times to obtain key pairs (SK_1, PK_1), $\ldots$, (SK_m, PK_m). It sets the secret key to $SK = (SK_1, \ldots, SK_m)$ and the public key to $PK = (PK_1, \ldots, PK_m)$.
- Signature: $Sign'(SK, M) = (Sign(SK_1, M_1), \ldots, Sign(SK_m, M_m)$ where M_i is the i'th bit of M.
- Verification: $Ver'(PK, M, T) = 1$ if $Ver(PK_i, M_i, T_i) = 1$ for every i, $1 \leq i \leq m$ and 0 otherwise, where M_i is the i'th bit of M and T_i it the i'th block of T.

It is easy to see that $(Gen', Sign', Ver')$ is a functional one-time digital signature for message length m. In Homework 3 you will also prove that it is secure:

Theorem 4. *Assume $(Gen, Sign, Ver)$ is an (s, ε) one-time secure bit signature. Then $(Gen', Sign', Ver')$ is an $(s - O(mt), \varepsilon/m)$ one-time secure signature scheme for m bit messages, where t is an upper bound on the size of Gen, $Sign$, and Ver.*

Signing messages of arbitrary length One issue with the above signature scheme is that the key size is quite a bit larger than the message length. This is not merely a practical limitation. In the next section where we will lift the one-time restriction on the signature scheme we will need to sign messages that are at least as long as the public key. Just like in the setting of MACs, we can apply cryptographic hash functions to turn a fixed-length MAC into one of arbitrary length.

Formally, let $(Gen, Sign, Ver)$ be a signature scheme for message length k and $\{h_S \colon \{0,1\}^* \to \{0,1\}^k\}$ be a cryptographic hash function family. We define the signature scheme $(Gen', Sign', Ver')$ for messages of arbitrary length:

- Key generation: Gen' runs Gen to generate a key pair (SK, PK) and chooses a random key S for h_S. It outputs the key pair (SK', PK') where $SK' = (SK, S)$ and $PK' = (PK, S)$.
- Signature: $Sign'(SK', M) = Sign(SK, h_S(M))$
- Verification: $Ver'(PK', M, T) = 1$ if $Ver(PK, h_S(M), T) = 1$ and 0 if not.

The security of $(Gen', Sign', Ver')$ is proved almost exactly like Claim 2 in Lecture 6, where we argued a similar claim for MACs. Actually this proof is a bit easier because we only need to assume and argue one-time security.

Theorem 5. *Assume $(Gen, Sign, Ver)$ is an (s, ε) one-time secure signature scheme for message length k and $\{h_S\}$ is an (s, ε) secure cryptographic hash family where h_S is computable by a circuit of size t. Then $(Gen', Sign', Ver')$ is a $(\Omega(s/t), 2\varepsilon)$ one-time secure signature scheme for arbitrary message length.*

3 Constructing digital signatures

It is easy to see that if more than one access to the signing oracle is allowed, a forgery can be produced even for a message of length 2: For example querying the signatures of the messages 00 and 11 and "cutting and pasting" the corresponding blocks we can forge the signature of 01.

We will however make use of the one-time signature in constructing signatures that are unforgeable without a restriction on the number of times they are used. Let's focus on constructing a signature scheme for an arbitrary but specific message length m. The idea will be to sign every message M under a *different* secret key-pubilc key pair (SK_M, PK_M).

How do we go about constructing an unforgeable signature scheme then? One idea is to use the one-time signature scheme, but with a different pair of keys (SK_M, PK_M) for every message M. If the key pairs are mutually independent, observing the signatures of messages $M_1, \ldots, M_k$ should not give us any information about what the signature of a message M different from the previous ones should look like because the key pair (SK_M, PK_M) is independent of $(SK_{M_1}, PK_{M_1}), \ldots, (SK_{M_k}, PK_{M_k})$.

The problem with this construction is that all together the keys are now over 2^m bits long and so the key generation algorithm is not efficient. So let us look into key generation in more detail. Recall that for every M, the key generation algorithm chooses some randomness $R(M) \in \{0,1\}^k$ and computes the key pair $(SK_M(R(M)), PK_M(R(M))$. We said it would be desirable for the key pairs (SK_M, PK_M) to be mutually independent, which means that the values $R(M)$ should be independent of one another. In other words, if we think of R as a function from $\{0,1\}^m$ to $\{0,1\}^k$

we would like it to be a random function. However, to describe this function Alice and Bob need $k2^m$ bits of information. What if we replace R with a pseudorandom function F_K which can now be described only by specifying the short key K?

Can such a scheme be implemented? Let $(Gen, Sign, Ver)$ be a one-time signature scheme. To sign a message M, Alice would output $T = Sign(SK_M, M)$ and to verify the signature, Bob would check that $Ver(PK_M, M, T) = 1$, where the key pair (SK_M, PK_M) is obtained by running Gen with randomness $F_K(M)$. This describes how signing and verifying signatures should work. What about key generation? If we include K as (part of) the secret key, Alice can generate SK_M by running Gen with randomness $F_K(M)$ at runtime when she needs to sign M.

But how does Bob now know what PK_M is supposed to be? What Alice can do is also send PK_M to Bob at runtime as part of the signature. So now to compute the signature of M, Alice would first run Gen with randomness $F_K(M)$ to obtain (SK_M, PK_M) and then sign M with the pair $(PK_M, Sign(SK_M, M))$. To verify this signature, Bob should now run the following procedure

$$Ver'(?, M, (PK_M, T)) = Ver(PK_M, M, T).$$

This construction looks suspicious as the verification Ver' makes no use of its public key! In fact forgeries are now quite easy to obtain even without a signing oracle (you should figure this out yourself). The problem is that in the absence of a public key, Bob has no proof that PK_M was indeed obtained by running Gen with randomness $F_K(M)$.

Here is the final ingredient of the construction: Alice will sign the public key PK_M itself under a master key (SK, PK). But this assumes we already have a signature scheme, which is something we wanted to construct in the first place! It turns out we can implement a signature for PK_M recursively by using a public-key variant of the "database commitment scheme" from Problem 4 on Homework 2.

The construction Let $(Gen, Sign, Ver)$ be a one-time signature scheme and $\{F_K \colon \{0,1\}^{\leq m} \to \{0,1\}^k\}$ be a one-way function family. Here $\{0,1\}^{\leq m}$ denotes all strings of length at most m and k is the number of random bits used by Gen. We construct the following signature scheme $(Gen', Sign', Ver')$:

- Key Generation: Gen' chooses a random seed K for F_K. It runs Gen with randomness $F_K(\varepsilon)$ (ε is the empty string) to obtain a master key pair $(SK_\varepsilon, PK_\varepsilon)$. It outputs the key pair (K, PK_ε).
- Signature: Given a secret key K and a message M, $Sign(K, M)$ does the following. Let M^i, where $0 \leq i \leq m$ denote the i-bit prefix of M. Output the sequence

$$\begin{aligned}
&(PK_0PK_1, Sign(SK_\varepsilon, PK_0PK_1),\\
&\ PK_{M^1 0}PK_{M^1 1}, Sign(SK_{M^1}, PK_{M^1 0}PK_{M^1 1}),\\
&\qquad \ldots,\\
&\ PK_{M^{m-1} 0}PK_{M^{m-1} 1}, Sign(SK_{M^{m-1}}, PK_{M_{m-1} 0}PK_{M_{m-1} 1}),\\
&\ Sign(SK_m, M))
\end{aligned}$$

  where (SK_Q, PK_Q) is the key pair obtained by running Gen with randomness $F_K(Q)$ with $Q \in \{0,1\}^{\leq m}$.

- Verification: Given a public key PK_ε, message M, and signature of the form

$$(PK_0PK_1, T_1, PK_{M^1 0}PK_{M^1 1}, T_2, \ldots, PK_{M^{m-1}0}PK_{M^{m-1}1}, T_m, T)$$

accept (output 1) if

$$\begin{aligned}
&Ver(PK_\varepsilon, PK_0PK_1, T_1) = 1 \quad \text{and} \\
&Ver(PK_{M^1}, PK_{M^1 0}PK_{M^1 1}, T_2) = 1 \quad \text{and} \\
&\qquad \ldots \\
&Ver(PK_{M^{m-1}}, PK_{M^{m-1}0}PK_{M^{m-1}1}, T_m) = 1 \quad \text{and} \\
&Ver(PK_M, M, T) = 1
\end{aligned}$$

and reject (output 0) otherwise.

Assuming $(Gen, Sign, Ver)$ is a functional signature scheme, the functionality of $(Gen', Sign', Ver')$ follows by definition. We now argue that this scheme is secure.

Theorem 6. *If $(Gen, Sign, Ver)$ is $(O(s'), \varepsilon'/2s')$ one-time secure and $\{F_K\}$ is an $(O(s't), \varepsilon'/2)$ pseudorandom function family, then $(Gen', Sign', Ver')$ is (s', ε') unforgeable, where t is an upper bound on the circuit size of Gen', $Sign'$, and Ver'.*

Instead of proving this theorem formally, which is a bit tedious, let's try to understand why this is true by working out a special case. First of all, let us pretend that the function F_K is a truly random function R. (Formally, we would do this by defining an ideal signature scheme $(RGen', RSign', RVer')$ that is the same as $(Gen', Sign', Ver')$ except that it uses R instead of F_K and its secret key is R instead of K and arguing that if the two can be distinguished by the adversary then F_K can be distinguished from a random function R.) This means the key pairs (SK_Q, PK_Q) are now completely independent of one another.

Now let's set $m = 2$ and suppose you have an adversary $A'^?$ that on input PK_ε and given oracle access to $Sign'(SK_\varepsilon, \cdot)$ outputs a forgery with some probability. To be specific, suppose $A'^{Sign'(SK_\varepsilon,\cdot)}(PK_\varepsilon)$ queries the oracle on messages 01 and 10 and manages to produced a forged signature of 11. In other words, the algorithm receives the query answers

$$\begin{aligned}
&PK_0PK_1, Sign(SK_\varepsilon, PK_0PK_1), PK_{00}PK_{01}, Sign(SK_0, PK_{00}PK_{01}), Sign(SK_{01}, 01) \quad \text{and} \\
&PK_0PK_1, Sign(SK_\varepsilon, PK_0PK_1), PK_{10}PK_{11}, Sign(SK_1, PK_{10}PK_{11}), Sign(SK_{10}, 10)
\end{aligned}$$

and outputs a forgery $11, (PK'_0PK'_1, T_1, PK'_{10}PK'_{11}, T_2, T)$ such that

$$Ver(PK_\varepsilon, PK'_0PK'_1, T_1) = Ver(PK'_1, PK'_{10}PK'_{11}, T_2) = Ver(PK'_{11}, 11, T) = 1.$$

If $PK'_0PK'_1 \neq PK_0PK_1$, then T_1 is a forged signature of $PK'_0PK'_1$ under the key pair $(SK_\varepsilon, PK_\varepsilon)$. To produce this forgery we would define an adversary $A^?$ that on input PK_ε and given oracle access to S, simulates $A'^?(PK_\varepsilon)$. When $A'^?$ makes its first query 01, A^S answers by

$$PK_0PK_1, S(PK_0PK_1), PK_{00}PK_{01}, Sign(SK_0, PK_{00}PK_{01}), Sign(SK_{01}, 01)$$

where (SK_Q, PK_Q) with $Q \in \{0, 1, 00, 01\}$ are freshly generated independent key pairs. When $A'^?$ makes its second query 10, A^S answers by

$$PK_0PK_1, S(PK_0PK_1), PK_{10}PK_{11}, Sign(SK_1, PK_{10}PK_{11}), Sign(SK_{10}, 10)$$

where SK_1, PK_0 and PK_1 are known from before, and (SK_Q, PK_Q) for $Q \in \{10, 11\}$ are freshly generated again. When $A'^?$ outputs its forgery, $A^?$ outputs $PK'_0PK'_1$. Notice that $A^?$ makes only one call to its signing oracle, so $(PK'_0PK'_1, T_1)$ is indeed a one-time forgery for the key pair $(SK_\varepsilon, PK_\varepsilon)$.

If $PK'_0PK'_1 = PK_0PK_1$ but $PK'_{10}PK'_{11} \neq PK_{10}PK_{11}$, then we can similarly argue that $PK'_{10}PK'_{11}$ is a forged signature under the key pair (SK_1, PK_1). Now the adversary $A^?$ will interpret its input at PK_1 and it will again simulate $A^?$ on input PK_ε where PK_ε as well as all other keys in the simulation are generated using fresh randomness. By a similar analysis we can conclude that $(PK'_{10}PK'_{11}, T_2)$ is a one-time forgery for the key pair (SK_1, PK_1).

If $PK'_0PK'_1 = PK_0PK_1$ and $PK'_{10}PK'_{11} = PK_{10}PK_{11}$, then $A'^?$ can be used to output a forgery of the message 11 under key PK_{11} without having seen any signature under SK_{11} before. Now $A^?$ interprets its input as PK_{11}, does a similar simulation as before and outputs $(11, T)$ as its forgery.