U.C. Berkeley — CS276: Cryptography Handout N20
Luca Trevisan April 2, 2009

Notes for Lecture 20

Scribed by Nick Jalbert, posted May 1, 2009

Summary

Today we begin to talk about *signature schemes.*

We describe various ways in which "textbook RSA" signatures are insecure, develop the notion of *existential unforgeability under chosen message attack*, analogous to the notion of security we gave for authentication, and discuss the difference between authentication in the private-key setting and signatures in the public-key setting.

As a first construction, we see Lamport's *one-time signatures* based on one-way functions, and we develop a rather absurdly inefficient stateful scheme based on one-time signatures. The scheme will be interesting for its idea of "refreshing keys" which will be used next time to design a stateless, and reasonably efficient, scheme.

1 Signature Schemes

Signatures are the public-key equivalents of MACs. The set-up is that Alice wants to send a message M to Bob, and convince Bob of the authenticity of the message. Alice generates a public-key/ secret-key pair (pk, sk), and makes the public key known to everybody (including Bob). She then uses an algorithm $Sign()$ to compute a *signature* $\sigma := Sign(sk, M)$ of the message; she sends the message along with the signature to Bob. Upon receiving M, σ, Bob runs a verification algorithm $Verify(pk, M, \sigma)$, which checks the validity of the signature. The security property that we have in mind, and that we shall formalize below, is that while a valid signature can be efficiently generated given the secret key (via the algorithm $Sign()$), a valid signature cannot be efficiently generated without knowing the secret key. Hence, when Bob receives a message M along with a signature σ such that $Verify(pk, M, \sigma)$ outputs "valid," then Bob can be confident that M is a message that came from Alice. (Or, at least, from a party that knows Alice's secret key.)

There are two major differences between signatures in a public key setting and MAC in a private key setting:

1. **Signatures are transferrable:** Bob can forward (M, σ) to another party and the other party can be confident that Alice actually sent the message, assuming

a public key infrastructure is in place (or, specifically that the other party has Alice's public key).

2. **Signature are non-repudiable:** Related to the first difference, once Alice signs the message and sends it out, she can no longer deny that it was actually her who sent the message.

Syntactically, a signature scheme is a collection of three algorithms $(Gen, Sign, Verify)$ such that

- $Gen()$ takes no input and generates a pair (pk, sk) where pk is a public key and sk is a secret key;
- Given a secret key sk and a message M, $Sign(sk, M)$ outputs a signature σ;
- Given a public key pk, a message M, and an alleged signature σ, $Verify(sk, M, \sigma)$ outputs either "valid" or "invalid", with the property that for every public key/ secret key pair (pk, sk), and every message M,

$$Verify(pk, M, Sign(sk, M)) = \text{ "valid"}$$

The notion of a signature scheme was described by Diffie and Hellman without a proposed implementation. The RSA paper suggested the following scheme:

- Key Generation: As in RSA, generate primes p, q, generate e, d such that $ed \equiv 1 \bmod (p-1)\cdot(q-1)$, define $N := p \cdot q$, and let $pk := (N, e)$ and $sk := (N, d)$.
- Sign: for a message $M \in \{0, \ldots, N-1\}$, the signature of M is $M^d \bmod N$
- Verify: for a message M and an alleged signature σ, we check that $\sigma^e \equiv M \pmod N$.

Unfortunately this proposal has several security flaws.

- Generating random messages with valid signatures: In this attack, you pick a random string σ and compute $M := \sigma^e \bmod N$ and now σ is a valid signature for M. This can be an effective attack if there are a large number of messages that are useful to forge.
- Combining signed messages to create the signature of their products: Suppose you have M_1 and M_2 with valid signatures σ_1 and σ_2 respectively. Note that $\sigma_1 := M_1^e \bmod N$ and $\sigma_2 := M_2^e \bmod N$. We can now generate a valid signature for $M := M_1 \cdot M_2 \bmod N$:

$$\begin{aligned}\sigma_M &= M^e \bmod N \\ &= (M_1 \cdot M_2)^e \bmod N \\ &= M_1^E \cdot M_2^e \bmod N \\ &= \sigma_1 \cdot \sigma_2 \bmod N\end{aligned}$$

- Creating signatures for arbitrary messages: Suppose the adversary wants to forge message M. If it is able to get a valid signature for a random message m_1 and a specifically chosen message $m_2 := M/m_1 \bmod N$ then the adversary can use the second attack to calculate a valid signature for M (i.e. $m_1 \cdot m_2 \bmod N = M$ and $\sigma_1 \cdot \sigma_2 \bmod N = \sigma_M$.

Ideally, we would like the following notion of security, analogous to the one we achieved in the secret-key setting.

Definition 1 *A signature scheme (G, S, V) is (t, ϵ) existentially unforgeable under a chosen message attack if for every algorithm A of complexity at most t, there is probability $\leq \epsilon$ that A, given a public key and a signing oracle, produces a valid signature of a message not previously sent to the signing oracle.*

It was initially thought no signature scheme could meet this definition. The so called "paradox" of signature schemes was that it seemed impossible to both have a scheme in which forgery is difficult (that is, equivalent to factoring) while simultaneously having this scheme be immune to chosen message attacks. Essentially the paradox is that the proof that a scheme is difficult to forge will generally use a black-box forging algorithm to then construct a factoring algorithm. However, if this scheme were subject to a chosen message attack, a new algorithm could be constructed which would simulate the constructed factoring algorithm and totally break the signature scheme. This new algorithm would be exactly the same as the one used in the forgery proof except every query to the black-box forging algorithm instead becomes one of the messages sent to the oracle. Goldwasser et al.'s paper "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks" gives further details and describes breaking the paradox.

2 One-Time Signatures and Key Refreshing

We begin by describing a simple scheme which achieves a much weaker notion of security.

Definition 2 (One-Time Signature) *A signature scheme (G, S, V) is a (t, ϵ)-secure one-time signature scheme if for every algorithm A of complexity at most t, there is probability $\leq \epsilon$ that A, given a public key and one-time access to a signing oracle, produces a valid signature of a message different from the one sent to the signing oracle.*

We describe a scheme due to Leslie Lamport that is based on one-way function.

- KeyGen G: pick a secret key which consists of 2ℓ n-bit random strings

  $$x_{0,1},\ x_{0,2}, ...,\ x_{0,\ell}$$

  $$x_{1,1},\ x_{1,2}, ...,\ x_{1,\ell}$$

  we now generate a public key by applying f to each $x_{*,*}$ in our secret key:

  $$f(x_{0,1}), f(x_{0,2}), ..., f(x_{0,\ell})$$
  $$f(x_{1,1}), f(x_{1,2}), ..., f(x_{1,\ell})$$

- Sign S: we sign an ℓ-bit message $M := M_1||M_2||...||M_\ell$ with the signature $\sigma := x_{m_1,1},\ x_{m_2,2}, ...,\ x_{m_\ell,\ell}$

  e.g. the message M:= 0110 gets the signature $\sigma_M := x_{0,1},\ x_{1,2},\ x_{1,3},\ x_{0,4}$

- Verify V: we verify a message $M := M_1...M_\ell$'s signature $\sigma := z_1...z_\ell$ by using public key pk and checking $\forall i\ f(z_i) = pk_{M_i,i}$

Theorem 3 *Let $f : \{0,1\}^n \to \{0,1\}^n$ be a (t, ϵ) one way function computable in time r. Then there is a one-time signature scheme (G, S, V) that signs messages of length ℓ and that is $(t - O(rl), \epsilon \cdot 2\ell)$ secure.*

Suppose this scheme does not provide a (t, ϵ)-one time signature. This implies that there must be an algorithm A with complexity $\leq t$ which makes 1 oracle query and

$$\mathbb{P}[A^{S(sk,\cdot)}(pk) \text{ forges a signature}] > \epsilon$$

Intuitively, when we are given a string $y := f(x)$ we want to use A to break the security of f and determines the pre-image of y.

We now describe the operation A', the algorithm that breaks the security of f. A' begins by generating a public key pk (which requires 2ℓ evaluations of f). Once A' generates pk, it sets a random position in it to the string y. Note the distribution of values in this modified pk will look exactly to the same A because y is also an evaluation of f.

A' now runs A passing it pk, A will query the oracle with a message to sign. With probability $1/2$ this message will not require A' to invert y. If this is the case, then with probability $> \epsilon$ A generates a forged message M' and signature $\sigma_{M'}$. M' must differ from the oracle query by at least one bit, that is this forgery finds the inverse of at least one element of pk not queried. This inverse will be y^{-1} with probability $1/\ell$.

A' runs in time at most $t + O(r\ell)$ if r is the running time of f and and inverts y with probability $\epsilon/2\ell$. Thus if we take f to be (t, ϵ)-secure, then this signature scheme must be $(t - O(rl), \epsilon \cdot 2\ell)$ secure. $\square$

A disadvantage of the scheme (besides the fact of being only a one-time signature scheme) is that the length of the signature and of the keys is much bigger than the length of the message: a message of length ℓ results in a signature of length $\ell \cdot n$, and the public key itself is of length $2 \cdot \ell \cdot n$.

Using a collision resistant hash function, however, we can convert a one-time signature scheme that works for short messages into a one-time signature scheme that works for longer messages. (Without significantly affecting key length, and without affecting the signature length at all.)

We use the hash function to hash the message into a string of the appropriate length and then sign the hash:

- $G'()$: generates a secret key and a public key (sk, pk) as G did above. Also picks a random seed $d \in \{0,1\}^k$ which becomes part of the public key.
- $S'(sk, M)$: $\sigma := S(sk, H_d(M))$
- $V'((pk, d), M, \sigma)$: $V(pk, H_d(M), \sigma)$

Theorem 4 *Suppose (G, S, V) is a (t, ϵ) secure one-time signature scheme for messages of length ℓ, which has public key length kl and signature length sl. Suppose also that we have a (t, ϵ) secure family of collision-resistant hash functions $H : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^\ell$. Suppose, finally, that H, G, S all have running time at most r.*

Then there exists a $(t - O(r), 2\epsilon)$ secure one-time signature scheme (G', S', V') with public key length $kl + k$, signature length sl and which can sign messages of length m.

We present a sketch of the proof of this theorem.

Suppose we had an algorithm A that produced forged signatures with probability $> 2\epsilon$ after one oracle query. That is, A queries the oracle with message M and gets back $\sigma := S'(sk, H_d(M))$ and then produces with probability $> 2\epsilon$ a message signature pair, (M', σ'), such that $V(pk, H_d(M'), \sigma') = \mathit{valid}$ and $M \neq M'$.

One of the two following cases must occur with probability $> \epsilon$:

1. The message M' has the same hash as M, $H(M) = H(M')$. This means A was able to find a collision in H. If A does this with probability $> \epsilon$ then it contradicts our security assumptions on H.

2. If $H(M) \neq H(M')$, then A forged a signature for the original scheme (G, S, V) for a fresh message. If A can do this with probability $> \epsilon$ then it contradicts our security assumptions on (G, S, V).

Because we reach a contradiction in both cases, (G', S', V') must be a $(t - O(r), 2\epsilon)$ secure one-time signature scheme.

In particular, given a one-way function and a family of collision-resistant hash functions we can construct a one-time signature scheme in which the length of a signature plus the length of the public key is less than the length of messages that can be signed by the scheme.

If (G, S, V) is such a one-time signature scheme, then the following is a stateful scheme that is existentially unforgeable under a chosen message attack.

Initially, the signing algorithm generates a public key/ secret key pair (pk, sk). When it needs to sign the first message M_1, it creates a new key pair (pk_1, sk_1), and generates the signature $\sigma_0 := S(sk, M_1||pk_1)$. The signature of M_1 is the pair (σ_0, pk_1). When it, later, signs message M_2, the signing algorithm generates a new key pair (pk_2, sk_2), and the signature $\sigma_1 = S(sk_1, M_2||pk_2)$. The signature of M_2 is the sequence

$$M_1, pk_1, \sigma_0, pk_2, \sigma_1$$

and so on. Of course it is rather absurd to have a signature scheme in which the signature of the 100th message contains in its entirety the *previously signed* 100 messages along with their signatures, but this scheme gives an example of the important paradigm of *key refreshing*, which will be more productively employed next time.