CPSC 467: Cryptography and Computer Security

Michael J. Fischer

Lecture 12
October 9, 2017

Using Digital Signatures

Adding redundancy

Recall: RSA signatures are subject to existential forgery.

Redundancy can be used to prevent this.

Example: Prefix a fixed string σ to the front of each message before signing.

This gives rise to a variant RSA signature scheme (S_d^σ, V_e^σ).

- Signing function: $S_d^\sigma(m) = D_d(\sigma m)$
- Verification predicate: $V_e^\sigma(m, s) \iff \sigma m = E_e(s)$.

Security of signatures with fixed redundancy

The security of this scheme depends on the mixing properties of the encryption and decryption functions, that is, the extent to which each output bit depends on most of the input bits.

Not all cryptosystems mix well.

For example, a block cipher used in ECB mode (see lecture 6 and lecture 8) encrypts a block at a time, so each block of output bits depends only on the corresponding block of input bits.

Forging signatures with fixed redundancy

Suppose it happens that

$$S_d^\sigma(m) = D_d(\sigma m) = D_d(\sigma) \cdot D_d(m).$$

Then Mallory can forge random messages assuming he knows just one valid signed message (m_0, s_0). Here's how.

- He knows that $s_0 = D_d(\sigma) \cdot D_d(m)$, so from s_0 he extracts the prefix $s_{00} = D_d(\sigma)$.
- He now chooses a random s'_{01} and computes $m' = E_e(s'_{01})$ and $s' = s_{00} \cdot s'_{01}$.
- The signed message (m', s') is valid since $E_e(s') = E_e(s_{00} \cdot s'_{01}) = E_e(s_{00}) \cdot E_e(s'_{01}) = \sigma m'$.

Signing Message Digests

Message digests

A better way to prevent forgery is to sign a *message digest* of the message rather than sign m itself.

A message digest function h, also called a *cryptographic one-way hash function* or a *fingerprint function*, maps long strings to short random-looking strings.

- To sign a message m, Alice computes $S_d(m) = D_d(h(m))$.
- To verify the signature s, Bob checks that $h(m) = E_e(s)$.

Forging signed message digests

For Mallory to generate a forged signed message (m', s') he must somehow come up with m' and s' satisfying

$$h(m') = E_e(s') \tag{1}$$

That is, he must find m' and s' that both map to the same string, where m' is mapped by h and s' by E_e.

Two natural approaches for attempting to satisfying (1):

1. Pick m' at random and solve for s'.
2. Pick s' at random and solve for m'.

Approach 1: Solve for s'

Equation:

$$h(m') = E_e(s') \tag{1}$$

To solve for s' given m' requires computing

$$E_e^{-1}(h(m')) = D_d(h(m')) = s'.$$

Alice can compute D_d, which is what enables her to sign messages.

But Mallory presumably cannot compute D_d without knowing d, for if he could, he could also break the underlying cryptosystem.

Approach 2: Solve for m'

Equation:

$$h(m') = E_e(s') \tag{1}$$

To solve for m' given s' requires "inverting" h.

Since h is many-one, a value $y = E_e(s')$ can have many "inverses" or *preimages*.

To successfully forge a signed message, Mallory needs to find only one value m' such that $h(m') = E_e(s')$.

However, the defining property of a cryptographic hash function is that, given y, it should be hard to find any $x \in h^{-1}(y)$.

Hence, Mallory cannot feasibly find m' satisfying 1.

Other attempts

Of course, these are not the only two approaches that Mallory might take.

Perhaps there are ways of generating valid signed messages (m', s') where m' and s' are generated together.

I do not know of such a method, but this doesn't say one doesn't exist.

More advantages of signing message digests

Another advantage of signing message digests rather than signing messages directly: the signatures are shorter.

An RSA signature of m is roughly the same length as m.

An RSA signature of $h(m)$ is a fixed length, regardless of how long m is.

For both reasons of security and efficiency, signed message digests are what is used in practice.

We'll talk more about message digests later on.

Signed encrypted messages

One often wants to encrypt messages for privacy and sign them for integrity and authenticity.

Let Alice have cryptosystem (E, D) and signature system (S, V). Some possibilities for encrypting and signing a message m:

1. Alice separately encrypts and signs the message and sends the pair $E(m) \circ S(m)$.
2. Alice signs the encrypted message and sends the pair $E(m) \circ S(E(m))$.
3. Alice encrypts the signed message and sends the result $E(m \circ S(m))$.

Here we assume a standard way of representing the ordered pair (x, y) as a string, which we denote by $x \circ y$.

Weakness of encrypt-and-sign

Method 1, sending the pair $E(m) \circ S(m)$, is quite problematic since *signature functions make no guarantee of privacy*.

We can construct a signature scheme (S', V') in which the plaintext message is part of the signature itself.

If (S', V') is used as the signature scheme in method 1, there is no privacy, for the plaintext message can be read directly from the signature.

A forgery-resistant signature scheme with no privacy

We construct a contrived but valid signature scheme in order to prove that not all signature schemes hide the message.

Let (S, V) be an RSA signature scheme. Define

$$S'(m) = m \circ S(m) \; ;$$

$$V'(m, s) = \exists t(s = m \circ t \wedge V(m, t)) \; .$$

Facts

- (S', V') is at least as secure as (S, V).
- S' leaks m.

Why? Suppose a forger produces a valid signed message (m, s) in (S', V'), so $s = m \circ t$ for some t and $V(m, t)$ holds..

Then (m, t) is a valid signed message in (S, V).

Encrypt first

Recall method 2 (encrypt first): (E(m), S(E(m))).

This allows Eve to verify that the signed message was sent by Alice, even though Eve cannot read it.

Whether or not this property is desirable is application-dependent.

More importantly, if a signature scheme such as RSA is used that allows forging valid signed random messages, then Mallory could forge a ciphertext c with Alice's valid signature $s = (c, S(c))$.

Bob, believing c is valid, might proceed to decrypt c and act on the resulting message m.

Sign first

Recall method 3 (sign first): $E(m \circ S(m))$.

This forces Bob to decrypt a bogus message before discovering that it wasn't sent by Alice.

This method also fails if Mallory can forge a valid signed random message (m, s), for Mallory can proceed to encrypt $m \circ s$ (using Bob's public encryption key) and the result looks like it was produced by Alice.

Combining protocols

Subtleties emerge when cryptographic protocols are combined, even in a simple case like this where it is only desired to combine privacy with authenticity.

Think about the pros and cons of other possibilities, such as sign-encrypt-sign, i.e., $(E(m \circ S(m)), S(E(m \circ S(m))))$.

Does it also fail with forged random signed messages?

Practical Signature Algorithms

ElGamal signature scheme

The *private signing key* consists of a primitive root g of a prime p and a random exponent x.

The *public verification key* consists of g, p, and a, where $a = g^x \bmod p$.

To sign m:

1. *Choose random $y \in \mathbf{Z}^*_{\phi(p)}$.*
2. *Compute $b = g^y \bmod p$.*
3. *Compute $c = (m - xb)y^{-1} \bmod \phi(p)$.*
4. *Signature $s = (b, c)$.*

To verify (m, s), where $s = (b, c)$:

1. *Check that $a^b b^c \equiv g^m \pmod p$.*

Why do ElGamal signatures work?

We have

$$a = g^x \bmod p$$

$$b = g^y \bmod p$$

$$c = (m - xb)y^{-1} \bmod \phi(p).$$

Want that $a^b b^c \equiv g^m \pmod{p}$. Substituting, we get

$$a^b b^c \equiv (g^x)^b (g^y)^c \equiv g^{xb+yc} \equiv g^m \pmod{p}$$

since $xb + yc \equiv m \pmod{\phi(p)}$.[1]

[1] Note the use of the identity from lecture 10, slide 34:
$u \equiv v \pmod{\phi(p)} \Leftrightarrow g^u \equiv g^v \pmod{p}$.

Digital signature standard

The commonly-used Digital Signature Algorithm (DSA) is a variant of ElGamal signatures. Also called the Digital Signature Standard (DSS), it is described in U.S. Federal Information Processing Standard FIPS 186–4.

It uses two primes: p, which is 1024 bits long,[2] and q, which is 160 bits long and satisfies $q \mid (p-1)$. Here's how to find them: Choose q first, then search for z such that $zq+1$ is prime and of the right length. Choose $p = zq+1$ for such a z.

[2] The original standard specified that the length L of p should be a multiple of 64 lying between 512 and 1024, and the length N of q should be 160. Revision 2, Change Notice 1 increased L to 1024. Revision 3 allows four (L, N) pairs: $(1024, 160)$, $(2048, 224)$, $(2048, 256)$, $(3072, 256)$.

DSA key generation

Given primes p and q of the right lengths such that $q \mid (p-1)$, here's how to generate a DSA key.

- Let $g = h^{(p-1)/q} \bmod p$ for any $h \in \mathbf{Z}_p^*$ for which $g > 1$. This ensures that $g \in \mathbf{Z}_p^*$ is a non-trivial q^{th} root of unity modulo p.
- Let $x \in \mathbf{Z}_q^*$.
- Let $a = g^x \bmod p$.

Private signing key: (p, q, g, x).

Public verification key: (p, q, g, a).

DSA signing and verification

Here's how signing and verification work:

To sign m:

1. *Choose random $y \in \mathbf{Z}_q^*$.*
2. *Compute $b = (g^y \bmod p) \bmod q$.*
3. *Compute $c = (m + xb)y^{-1} \bmod q$.*
4. *Output signature $s = (b, c)$.*

To verify (m, s), where $s = (b, c)$:

1. *Verify that $b, c \in \mathbf{Z}_q^*$; reject if not.*
2. *Compute $u_1 = mc^{-1} \bmod q$.*
3. *Compute $u_2 = bc^{-1} \bmod q$.*
4. *Compute $v = (g^{u_1} a^{u_2} \bmod p) \bmod q$.*
5. *Check $v = b$.*

Why DSA works

To see why this works, note that since $g^q \equiv 1 \pmod{p}$, then

$$r \equiv s \pmod{q} \quad \text{implies} \quad g^r \equiv g^s \pmod{p}.$$

This follows from the fact that g is a q^{th} root of unity modulo p, so $g^{r+uq} \equiv g^r (g^q)^u \equiv g^r \pmod{p}$ for any u.
Hence,

$$g^{u_1} a^{u_2} \equiv g^{mc^{-1}+xbc^{-1}} \equiv g^y \pmod{p}. \tag{2}$$

$$g^{u_1} a^{u_2} \bmod p = g^y \bmod p \tag{3}$$

$$v = (g^{u_1} a^{u_2} \bmod p) \bmod q = (g^y \bmod p) \bmod q = b$$

as desired. (Notice the shift between *equivalence* modulo p in equation 2 and *equality of remainders* modulo p in equation 3.)

Double remaindering

DSA uses the technique of computing a number modulo p and then modulo q.

Call this function $f_{p,q}(n) = (n \bmod p) \bmod q$.

$f_{p,q}(n)$ is not the same as $n \bmod r$ for any modulus r, nor is it the same as $f_{q,p}(n) = (n \bmod q) \bmod p$.

Example mod 29 mod 7

To understand better what's going on, let's look at an example. Take $p = 29$ and $q = 7$. Then $7|(29 - 1)$, so this is a valid DSA prime pair. The table below lists the first 29 values of $y = f_{29,7}(n)$:

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13
y	0	1	2	3	4	5	6	0	1	2	3	4	5	6

n	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
y	0	1	2	3	4	5	6	0	1	2	3	4	5	6	0

The sequence of function values repeats after this point with a period of length 29. Note that it both begins and ends with 0, so there is a double 0 every 29 values. That behavior cannot occur modulo any number r. That behavior is also different from $f_{7,29}(n)$, which is equal to n mod 7 and has period 7. (Why?)