CPSC 467: Cryptography and Computer Security

Michael J. Fischer

Lecture 10
September 30, 2013

Message Integrity and Authenticity

- Message authentication codes
- Asymmetric digital signatures
- Implications of Digital Signatures

Digital Signature Algorithms

- RSA digital signatures
- Signatures from non-commutative cryptosystems
- ElGamal digital signature scheme

Security of Digital Signatures

- Desired security properties
- Random signed messages
- Adding redundancy
- Signing message digests

Message Integrity and Authenticity

Protecting messages

Encryption protects message confidentiality.

We also wish to protect message integrity and authenticity.

- *Integrity* means that the message has not been altered.
- *Authenticity* means that the message is genuine.

The two are closely linked. The result of a modification attack by an active adversary could be a message that fails either integrity or authenticity checks (or both).

In addition, it should not be possible for an adversary to come up with a forged message that satisfies both integrity and authenticity.

Protecting integrity and authenticity

Authenticity is protected using symmetric or asymmetric digital signatures.

A *digital signature* (or MAC) is a string s that binds an individual or other entity A with a message m.

The recipient of the message *verifies* that s is a *valid signature* of A for message m.

It should hard for an adversary to create a valid signature s' for a message m' without knowledge of A's secret information.

This also protects integrity, since a modified message m' will not likely verify with signature s (or else (m', s) would be a successful forgery).

Message authentication codes (MACs)

A *Message Authentication Code* or *MAC* is a digital signature associated with a *symmetric (one-key) signature scheme*.

A MAC is generated by a function $C_k(m)$ that can be computed by anyone knowing the secret key k.

It should be hard for an attacker, without knowing k, to find any pair (m, ξ) such that $\xi = C_k(m)$.

This should remain true even if the attacker knows a set of valid MAC pairs $\{(m_1, \xi_1), \ldots, (m_t, \xi_t)\}$ so long as m itself is not the message in one of the known pairs.

Creating an authenticated message

Alice has a secret key k.

- Alice protects a message m (encrypted or not) by attaching a MAC $\xi = C_k(m)$ to the message m.
- The pair (m, ξ) is an *authenticated message*.
- To produce a MAC requires possession of the secret key k.

Verifying an authenticated message

Bob receives an authenticated message (m', ξ'). We assume Bob also knows k.

- Bob verifies the message's integrity and authenticity by verifying that $\xi' = C_k(m')$.
- If his check succeeds, he *accepts* m' as a valid message from Alice.
- To verify a MAC requires possession of the secret key k.

Assuming Alice and Bob are the only parties who share k, then Bob knows that m' came from Alice.

Cheating

Mallory *successfully cheats* if Bob accepts a message m' as valid that Alice never sent.

Assuming a secure MAC scheme, Mallory can not cheat with non-negligible success probability, even knowing a set of valid message-MAC pairs previously sent by Alice.

If he could, he would be able to construct valid forged authenticated messages, violating the assumed properties of a MAC.

Computing a MAC

A block cipher such as AES can be used to compute a MAC by making use of CBC or CFB ciphertext chaining modes.

In these modes, the last ciphertext block c_t depends on all t message blocks $m_1, \ldots, m_t$, so we define

$$C_k(m) = c_t.$$

Note that the MAC is only a single block long. This is in general much shorter than the message.

A MAC acts like a checksum for preserving data integrity, but it has the advantage that an adversary cannot compute a valid MAC for an altered message.

Protecting both privacy and authenticity

If Alice wants both privacy and authenticity, she can encrypt m and use the MAC to protect the ciphertext from alteration.

- Alice sends $c = E_k(m)$ and $\xi = C_k(c)$.
- Bob, after receiving c' and ξ', only decrypts c' after first verifying that $\xi' = C_k(c')$.
- If it verifies, then Bob assumes c' was produced by Alice, so he also assume that $m' = D_k(c')$ is Alice's message m.

Another possible use of a MAC

Another possibility is for Alice to send $c = E_k(m)$ and $\xi = C_k(m)$. Here, the MAC is computed from m, not c.

Bob, upon receiving c' and ξ', first decrypts c' to get m' and then checks that $\xi' = C_k(m')$, i.e., Bob checks $\xi' = C_k(D_k(c'))$

Does this work just as well?

In practice, this might also work, but its security *does not follow* from the assumed security property of the MAC.

The problem

The MAC property says Mallory cannot produce a pair (m', ξ') for an m' that Alice never sent.

It does *not* follow that he cannot produce a pair (c', ξ') that Bob will accept as valid, even though c' is not the encryption of one of Alice's messages.

If Mallory succeeds in convincing Bob to accept (c', ξ'), then Bob will decrypt c' to get $m' = D_k(c')$ and incorrectly accept m' as coming from Alice.

Example of a flawed use of a MAC

Here's how Mallory might find (c', ξ') such that $\xi' = C_k(D_k(c'))$.

Suppose the MAC function C_k is derived from underlying block encryption function E_k using the CBC or CFB chaining modes as described earlier, and Alice also encrypts messages using E_k with the same chaining rule.

Then the MAC is just the last ciphertext block c'_t, and Bob will always accept (c', c'_t) as valid.

Asymmetric digital signatures

An asymmetric (public-key) digital signature can be viewed as a 2-key MAC, just as an asymmetric (public-key) cryptosystem is a 2-key version of a classical cryptosystem.

In the literature, the term *digital signature* generally refers to the asymmetric version.

Asymmetric digital signatures

Let $\mathcal{M}$ be a *message space* and $\mathcal{S}$ a *signature space*.

A *signature scheme* consists of a private *signing key* d, a public *verification key* e, a *signature function* $S_d : \mathcal{M} \to \mathcal{S}$, and a *verification predicate* $V_e \subseteq \mathcal{M} \times \mathcal{S}$.[1]

A *signed message* is a pair $(m, s) \in \mathcal{M} \times \mathcal{S}$. A signed message is *valid* if $V_e(m, s)$ holds, and we say that (m, s) is *signed with* e.

[1] As with RSA, we denote the private component of the key pair by the letter d and the public component by the letter e, although they no longer have same mnemonic significance.

Fundamental property of a signature scheme

Basic requirement:

The signing function always produces a valid signature, that is,

$$V_e(m, S_d(m)) \tag{1}$$

holds for all $m \in \mathcal{M}$.

Assuming e is Alice's public verification key, and only Alice knows the corresponding signing key d, then a signed message (m, s) that is valid under e identifies Alice with m (possibly erroneously, as we shall see).

What does a digital signature imply?

We like to think of a digital signature as a digital analog to a conventional signature.

- A conventional signature binds a person to a document. Barring forgery, a valid signature indicates that a particular individual performed the action of signing the document.
- A digital signature binds a signing key to a document. Barring forgery, a valid digital signature indicates that a particular signing key was used to sign the document.

However, there is an important difference. A digital signature only binds the signing key to the document.

Other considerations must be used to bind the individual to the signing key.

Disavowal

An individual can always disavow a signature on the grounds that the private signing key has become compromised.

Here are two ways that this can happen.

- Her signing key might be copied, perhaps by keystroke monitors or other forms of spyware that might have infected her computer, or a stick memory or laptop containing the key might be stolen.
- She might deliberately publish her signing key in order to relinquish responsibility for documents signed by it.

For both of these reasons, one cannot conclude without a reasonable doubt that a digitally signed document was indeed signed by the purported holder of the signing key.

Practical usefulness of digital signatures

This isn't to say that digital signatures aren't useful; only that they have significantly different properties than conventional signatures.

In particular, they are subject to disavowal by the signer in a way that conventional signatures are not.

Nevertheless, they are still very useful in situations where disavowal is not a problem.

Digital Signature Algorithms

RSA digital signature scheme

RSA can be used for digital signatures as follows:

- Alice generates an RSA modulus n and key pair (e, d), where e is public and d private as usual.
- Let $S_d(m) = D_d(m)$, and let $V_e(m, s)$ hold iff $m = E_e(s)$.
- Must verify that $V_e(m, S_d(m))$ hold for all messages m, i.e., must check that $m = E_e(D_d(m))$ holds.
- This is the reverse of the condition we required for RSA to be a valid cryptosystem, viz. $D_d(E_e(m))$ for all $m \in \mathbf{Z}_m$.
- RSA satisfies both conditions since

$$m \equiv D_d(E_e(m)) \equiv (m^e)^d \equiv (m^d)^e \equiv E_e(D_d(m)) \pmod{n}.$$

Commutative cryptosystems

A cryptosystem with this property that $D_d \circ E_e = E_e \circ D_d$ is said to be *commutative*, where "$\circ$" denotes functional composition.

Indeed, any commutative public key cryptosystem can be used for digital signatures in exactly this same way as we did for RSA.

Signatures from non-commutative cryptosystems

We digress slightly and ask what we could do in case E_e and D_d did not commute.

One could define $S_e(m) = E_e(m)$ and $V_e(m, s) \Leftrightarrow m = D_d(s)$. Now indeed every validly-signed message $(m, S_e(m))$ would verify since $D_d(E_e(m)) = m$ is the basic property of a cryptosystem.

To make use of this scheme, Alice would have to keep e private and make d public. Assuming Alice generated the key pair in the first place, there is nothing preventing her from doing this. However, the resulting system might not be secure.

Even if it is hard for Eve to find d from e, it might not be hard to find e from d.

Interchanging public and private keys

For RSA, it is just as hard to find e from d as it is to find d from e. That's because RSA is completely symmetric in e and d.
Not all cryptosystems enjoy this symmetry property.

For example, the ElGamal scheme discussed in Lecture 9 is based on the equation $b = g^y \pmod{p}$, where y is private and b public.

Finding y from b is the discrete log problem — believed to be hard.

Finding b from y, is straightforward, so the roles of public and private key cannot be interchanged while preserving security.[2]

[2] However, ElGamal found a different way to use the ideas of discrete logarithm to build a signature scheme, which we discuss next.

ElGamal signature scheme

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

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

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. *Output 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 $xb + yc \equiv m \pmod{\phi(p)}$.

Security of Digital Signatures

Desired security properties of digital signatures

Digital signatures must be difficult to forge.

Some increasingly stringent notions of forgery-resistance:

- Resistance to forging valid signature for particular message m.
- Above, but where adversary knows a set of valid signed messages $(m_1, s_1), \ldots, (m_k, s_k)$, and $m \notin \{m_1, \ldots, m_k\}$.
- Above, but where adversary can choose a set of valid signed messages, specifying either the messages (corresponding to a chosen plaintext attack) or the signatures (corresponding to chosen ciphertext attack).
- Any of the above, but where one wishes to protect against generating any valid signed message (m', s') different from those already seen, not just for a particular predetermined m.

Forging random RSA signed messages

RSA signatures are indeed vulnerable to forgery of random signed messages.

An attacker simply chooses s' at random and computes $m' = E_e(s')$.

The signed message (m', s') is trivially valid since the verification predicate is simply $m' = E_e(s')$.

Importance of random signed messages

One often wants to sign random strings.

For example, in the Diffie-Hellman key exchange protocol discussed in Lecture 9, Alice and Bob exchange random-looking numbers $a = g^x \bmod p$ and $b = g^y \bmod p$.

In order to discourage man-in-the-middle attacks, they may wish to sign these strings. (This assumes that they already have each other's public signature verification keys.)

If RSA signatures are being used, Mallory could feed bogus signed values to Alice and Bob. The signatures would check, and both would think they had successfully established a shared key k when in fact they had not.

Adding redundancy

One way to defeat the adversary's ability to generate valid random signed messages is to put redundancy into the message, for example, by prefixing a fixed string σ to the front of each message before signing it.

Instead of taking $S_d(m) = D_d(m)$, one could take

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

The corresponding verification predicate would then be

$$V_e(m, s) \Leftrightarrow \sigma m = E_e(s).$$

Security of signatures with fixed redundancy

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

Not all cryptosystems have this mixing property.

For example, a block cipher used in ECB mode (see lectures 3 and 6) 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

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

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

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

$$h(m') = E_e(s') \tag{2}$$

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 (2):

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

Solving for s'

Approach 1:

$$h(m') = E_e(s') \tag{2}$$

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.

Solving for m'

Approach 2:

$$h(m') = E_e(s') \tag{2}$$

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 2.

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.