# CS 302 Computer Security and Privacy

Debayan Gupta

Lecture 6 Part 1
February 12, 2019

## ElGamal Cryptosystem

## Message Integrity and Authenticity

Message authentication codes
Asymmetric digital signatures
Implications of Digital Signatures

## Digital Signature Algorithms

Signatures from commutative cryptosystems
Signatures from non-commutative cryptosystems

## Security of Digital Signatures

Forgery

# ElGamal Cryptosystem

## A variant of DH key exchange

A variant protocol has Bob going first followed by Alice.

| **Alice** | **Bob** |
|---|---|
| | Choose random $y \in \mathbf{Z}_{\phi(p)}$. |
| | $b = g^y \bmod p$. |
| | Send $b$ to Alice. |
| Choose random $x \in \mathbf{Z}_{\phi(p)}$. | |
| $a = g^x \bmod p$. | |
| Send $a$ to Bob. | |
| $k_a = b^x \bmod p$. | $k_b = a^y \bmod p$. |

ElGamal Variant of Diffie-Hellman Key Exchange.

## Comparison with first DH protocol

The difference here is that Bob completes his action at the beginning and no longer has to communicate with Alice.

Alice, at a later time, can complete her half of the protocol and send *a* to Bob, at which point Alice and Bob share a key.

# Turning D-H into a public key cryptosystem

This is just the scenario we want for public key cryptography. Bob generates a public key $(p, g, b)$ and a private key $(p, g, y)$.

Alice (or anyone who obtains Bob's public key) can complete the protocol by sending $a$ to Bob.

This is the idea behind the ElGamal public key cryptosystem.

# ElGamal cryptosystem

Assume Alice knows Bob's public key $(p, g, b)$. To encrypt a message $m$:

- She first completes her part of the key exchange protocol to obtain numbers $a$ and $k$.
- She then computes $c = mk \bmod p$ and sends the pair $(a, c)$ to Bob.
- When Bob gets this message, he first uses $a$ to complete his part of the protocol and obtain $k$.
- He then computes $m = k^{-1}c \bmod p$.

# Combining key exchange with underlying cryptosystem

The ElGamal cryptosystem uses the simple encryption function $E_k(m) = mk \bmod p$ to actually encode the message.

Any symmetric cryptosystem would work equally well.

An advantage of using a standard system such as AES is that long messages can be sent following only a single key exchange.

## A hybrid ElGamal cryptosystem

A hybrid ElGamal public key cryptosystem.

- As before, Bob generates a public key $(p, g, b)$ and a private key $(p, g, y)$.
- To encrypt a message $m$ to Bob, Alice first obtains Bob's public key and chooses a random $x \in \mathbf{Z}_{\phi(p)}$.
- She next computes $a = g^x \bmod p$ and $k = b^x \bmod p$.
- She then computes $E_{(p,g,b)}(m) = (a, \hat{E}_k(m))$ and sends it to Bob. Here, $\hat{E}$ is the encryption function of the underlying symmetric cryptosystem.
- Bob receives a pair $(a, c)$.
- To decrypt, Bob computes $k = a^y \bmod p$ and then computes $m = \hat{D}_k(c)$.

# Randomized encryption

We remark that a new element has been snuck in here. The ElGamal cryptosystem and its variants require Alice to generate a random number which is then used in the course of encryption.

Thus, the resulting encryption function is a *random function* rather than an ordinary function.

A random function is one that can return different values each time it is called, even for the same arguments.

Formally, we view a random function as returning a probability distribution on the output space.

# Remarks about randomized encryption

With $E_{(p,g,b)}(m)$ each message $m$ has many different possible encryptions. This has some consequences.

**An advantage:** Eve can no longer use the public encryption function to check a possible decryption.

Even if she knows $m$, she cannot verify $m$ is the correct decryption of $(a, c)$ simply by computing $E_{(p,g,b)}(m)$, which she could do for a deterministic cryptosystem such as RSA.

**Two disadvantages:**

- Alice must have a source of randomness.
- The ciphertext is longer than the corresponding plaintext.

# 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 $(m, \xi)$ 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 $(m, \xi)$ is an *authenticated message*.
- To produce a MAC requires possession of the secret key $k$.

# Verifying an authenticated message

Bob receives an authenticated message $(m', \xi')$. 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 $\xi'$, only decrypts $c'$ after first verifying that $\xi' = C_k(c')$.
- If it verifies, then Bob assumes $c'$ was produced by Alice, so $m' = D_k(c')$ 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 $\xi'$, 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 $(m', \xi')$ for an $m'$ that Alice never sent.

It does *not* follow that he cannot produce a pair $(c', \xi')$ 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 $(c', \xi')$, 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 $(c', \xi')$ 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 respect to* $e$.

[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

Let $n$ be an RSA modulus and $(e, d)$ an RSA key pair.
$e$ is public and $d$ is private as usual.

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

Must verify that $V_e(m, S_d(m))$ holds for all messages $m$, i.e., that $m = E_e(D_d(m))$ holds.

This is the reverse of the requirement for RSA to be a valid cryptosystem, viz. $m = D_d(E_e(m))$ 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

What if $E_e$ and $D_d$ do not commute?

One could define:

- Signing function: $S_e(m) = E_e(m)$
- Verification predicate: $V_d(m, s) \Leftrightarrow m = D_d(s)$.

Every validly-signed message $(m, S_e(m))$ would verify since $D_d(E_e(m)) = m$ is the basic property of a cryptosystem.

Now, Alice has to keep $e$ private and make $d$ public, which she can do. However, the resulting system might not be secure, since even though it may be hard for Eve to find $d$ from $e$ and $n$, it does not follow that it is hard to find $e$ from $d$ and $n$.

# 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.

# ElGamal cryptosystem is not symmetric

The ElGamal scheme is based on the equation

$$b = g^y \pmod{p},$$

where $y$ is private and $b$ public.

Finding $y$ from $b, g, p$ is the discrete log problem — believed to be hard.

Finding $b$ from $y, g, p$, is straightforward, so the roles of public and private key cannot be interchanged while preserving security.

ElGamal found a different way to use the ideas of discrete logarithm to build a signature scheme, which we discuss later.

# 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 $m$. This is called *existential forgery*.

# Forging random RSA signed messages

RSA signatures are indeed vulnerable to existential forgery.

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, 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 (assuming they already have each other's public verification keys).

With RSA signatures, 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.