Digital Signatures

Valtteri Niemi

(text also from Timo Karvi and Ari Renvall)

Department of Computer Science
University of Helsinki, Finland

Fall 2017
Arguably the most useful application of public key cryptography is the concept of a digital signature. A digital signature scheme provides a method for a party to sign digital documents so that the signatures can later be verified by anyone. The scheme should guarantee

- **authenticity**, so that signatures cannot be forged;
- **integrity**, so that the contents of a signed message cannot be later altered; and
- **non-repudiation**, so that the signer cannot later deny having signed the message.
The idea behind digital signatures is quite similar to that of public-key cryptosystems. Alice has:

- a private signing key $s_A$
- a public verification key $v_A$.

Alice's signature $\text{sig}_A(m)$ for a message $m$ is then computed by $\text{sig}_A(m) = S(s_A, m)$, where $S$ is a publicly known signing algorithm.
Verifying the signature

Bob, or anyone who knows Alice’s public key, can verify the validity of Alice’s signatures using a publicly known verification algorithm $V$. It takes as input

- the message $m$,
- Alice’s claimed signature $s$ for $m$
- Alice’s public verification key $v_A$

and it computes a binary decision (Yes / No) whether $s = \text{sig}_A(m)$. 
A signature scheme is **secure** if it is **computationally infeasible** to forge signatures.

In other words, it should be computationally impossible to compute \( S(s_A, m) \) for any message \( m \) without the knowledge of \( s_A \).

This holds even in case where signatures for many messages \( m_i \) are known, i.e. if \( S(s_A, m_i) \) is known (for \( i = 1, \ldots, k \)) then it is infeasible to create a signature \( S'(s_A, m') \) for any \( m' \neq m_i \) (for \( i = 1, \ldots, k \)).
RSA signatures

Some public-key cryptosystems, for example RSA, can be used as digital signature schemes.
Let \((n, e, d)\) be Alice’s RSA key, and suppose Alice needs to sign a document \(m \in \mathbb{Z}_n\).
Alice signs \(m\) by “decrypting” it with her private key:

\[
s = \text{sig}_A(m) = m^d \pmod{n}.
\]

As only Alice knows \(d\), nobody can forge her signature for \(m\) (unless the RSA cryptosystem is insecure).
Bob can verify Alice’s claimed signature \(s\) for \(m\) by “encrypting” it with Alice’s public key. If \(s^e = m \pmod{n}\), then Bob is convinced that indeed \(s\) is Alice’s signature for \(m\), and that only Alice could have produced it.
Some issues

There are at least the following issues with digital signatures:

- How to sign longer messages?
  - In RSA: \( n \) cannot be too long and \( m < n \).

- In RSA anybody can create signatures for random messages:
  - Given a signature \( s \), how do you find a message \( x \) for which \( s \) is a valid signature?

- Answer: \( x = s^e \pmod{n} \).

- What if Alice denies that the public verifying key is hers?
Some issues

There are at least the following issues with digital signatures:

- How to sign longer messages?
  In RSA: \( n \) cannot be too long and \( m < n \).

- In RSA anybody can create signatures for random messages:
  Given a signature \( s \), how do you find a message \( x \) for which \( s \) is a valid signature?

  Answer: \( x = s^e \pmod n \).

- What if Alice denies that the public verifying key is hers?
Some issues

There are at least the following issues with digital signatures:

- How to sign longer messages?
  In RSA: $n$ cannot be too long and $m < n$.

- In RSA anybody can create signatures for random messages:
  Given a signature $s$, how do you find a message $x$ for which $s$ is a valid signature?
  
  **Answer:** $x = s^e \pmod{n}$.

- What if Alice denies that the public verifying key is hers?
Some issues

There are at least the following issues with digital signatures:

- How to sign longer messages?
  In RSA: $n$ cannot be too long and $m < n$.
- In RSA anybody can create signatures for random messages:
  Given a signature $s$, how do you find a message $x$ for which $s$ is a valid signature?
- Answer: $x = s^e \pmod{n}$.
- What if Alice denies that the public verifying key is hers?
Hash functions

The first two issues can be solved by using hash functions

- Instead of signing an arbitrary message \( m \) of arbitrary length, we sign a fixed length representative of \( m \), denoted by \( h(m) \)
- \( h \) is a hash function from \( \{0, 1\}^* \) to \( \{0, 1\}^k \) where \( k \) is a constant, e.g. \( k = 256 \).
- In RSA, the signer presents a message \( m \) with a signature \( s = h(m)^d \pmod{n} \).
- How to verify \((m, s)\)?
- Verifier computes \( h(m) \) and \( s^e \pmod{n} \). If the two results are the same then the signature is accepted.
The first two issues can be solved by using hash functions

- Instead of signing an arbitrary message $m$ of arbitrary length, we sign a fixed length representative of $m$, denoted by $h(m)$.

- $h$ is a hash function from $\{0, 1\}^*$ to $\{0, 1\}^k$ where $k$ is a constant, e.g. $k = 256$.

- In RSA, the signer presents a message $m$ with a signature $s = h(m)^d \pmod{n}$.

- How to verify $(m, s)$?

- Verifier computes $h(m)$ and $s^e \pmod{n}$. If the two results are the same then the signature is accepted.
The first two issues can be solved by using hash functions

- Instead of signing an arbitrary message $m$ of arbitrary length, we sign a fixed length representative of $m$, denoted by $h(m)$.
- $h$ is a hash function from $\{0, 1\}^*$ to $\{0, 1\}^k$ where $k$ is a constant, e.g. $k = 256$.
- In RSA, the signer presents a message $m$ with a signature $s = h(m)^d \pmod{n}$.

How to verify $(m, s)$?

Verifier computes $h(m)$ and $s^e \pmod{n}$. If the two results are the same then the signature is accepted.
Hash functions

The first two issues can be solved by using hash functions

- Instead of signing an arbitrary message $m$ of arbitrary length, we sign a fixed length representative of $m$, denoted by $h(m)$.
- $h$ is a hash function from $\{0, 1\}^*$ to $\{0, 1\}^k$ where $k$ is a constant, e.g. $k = 256$.
- In RSA, the signer presents a message $m$ with a signature $s = h(m)^d \pmod{n}$.
- How to verify $(m, s)$?
  - Verifier computes $h(m)$ and $s^e \pmod{n}$. If the two results are the same then the signature is accepted.
The first two issues can be solved by using hash functions:

- Instead of signing an arbitrary message $m$ of arbitrary length, we sign a fixed length representative of $m$, denoted by $h(m)$.
- $h$ is a hash function from $\{0, 1\}^*$ to $\{0, 1\}^k$ where $k$ is a constant, e.g. $k = 256$.
- In RSA, the signer presents a message $m$ with a signature $s = h(m)^d \pmod{n}$.
- How to verify $(m, s)$?
- Verifier computes $h(m)$ and $s^e \pmod{n}$. If the two results are the same then the signature is accepted.
A cryptographic hash function needs to have the following properties:

- **Compression:**
  \( h(m) \) has a fixed length \( k \) (e.g. 256 bits) while \( m \) may be of any length (also less than \( k \));

- **Efficiency:**
  \( h(m) \) is easy to compute (when \( m \) is known).

- **One-way:**
  Given a \( k \)-bit string \( h \), it is computationally infeasible to find any \( m \) such that \( h(m) = h \).

- **2nd-preimage resistance:**
  Given \( m \) it is computationally infeasible to find any \( m' \neq m \) such that \( h(m) = h(m') \).

- **Collision resistance:**
  It is computationally infeasible to find any two distinct \( m \) and \( m' \) such that \( h(m) = h(m') \).
Properties of hash functions

A cryptographic hash function needs to have the following properties

- **Compression:**
  \[ h(m) \] has a fixed length \( k \) (e.g. 256 bits) while \( m \) may be of any length (also less than \( k \));

- **Efficiency:**
  \[ h(m) \] is easy to compute (when \( m \) is known).

- **One-way:**
  Given a \( k \)-bit string \( h \), it is computationally infeasible to find any \( m \) such that \( h(m) = h \).

- **2nd-preimage resistance:**
  Given \( m \) it is computationally infeasible to find any \( m' \neq m \) such that \( h(m) = h(m') \).

- **Collision resistance:**
  It is computationally infeasible to find any two distinct \( m \) and \( m' \) such that \( h(m) = h(m') \).
Properties of hash functions

A cryptographic hash function needs to have the following properties:

- **Compression:**
  \( h(m) \) has a fixed length \( k \) (e.g. 256 bits) while \( m \) may be of any length (also less than \( k \));

- **Efficiency:**
  \( h(m) \) is easy to compute (when \( m \) is known).

- **One-way:**
  Given a \( k \)-bit string \( h \), it is computationally infeasible to find any \( m \) such that \( h(m) = h \).

- **2nd-preimage resistance:**
  Given \( m \) it is computationally infeasible to find any \( m' \neq m \) such that \( h(m) = h(m') \).

- **Collision resistance:**
  It is computationally infeasible to find any two distinct \( m \) and \( m' \) such that \( h(m) = h(m') \).
A cryptographic hash function needs to have the following properties:

- **Compression:**
  \( h(m) \) has a fixed length \( k \) (e.g. 256 bits) while \( m \) may be of any length (also less than \( k \));

- **Efficiency:**
  \( h(m) \) is easy to compute (when \( m \) is known).

- **One-way:**
  Given a \( k \)-bit string \( h \), it is computationally infeasible to find any \( m \) such that \( h(m) = h \).

- **2nd-preimage resistance:**
  Given \( m \) it is computationally infeasible to find any \( m' \neq m \) such that \( h(m) = h(m') \).

- **Collision resistance:**
  It is computationally infeasible to find any two distinct \( m \) and \( m' \) such that \( h(m) = h(m') \).
Properties of hash functions

A **cryptographic** hash function needs to have the following properties

- **Compression:**
  \( h(m) \) has a fixed length \( k \) (e.g. 256 bits) while \( m \) may be of any length (also less than \( k \));

- **Efficiency:**
  \( h(m) \) is easy to compute (when \( m \) is known).

- **One-way:**
  Given a \( k \)-bit string \( h \), it is **computationally infeasible** to find any \( m \) such that \( h(m) = h \).

- **2nd-preimage resistance:**
  Given \( m \) it is computationally infeasible to find any \( m' \neq m \) such that \( h(m) = h(m') \).

- **Collision resistance:**
  It is computationally infeasible to find any two distinct \( m \) and \( m' \) such that \( h(m) = h(m') \).
The message blocks $m_i$ are used as *keys* to the block cipher $E$.

The length of the hash is at most length of the block (in the cipher).

Finalization is just a technical thing, e.g. truncates to correct length.

The last message block $m_t$ includes padding.
One variant of the birthday paradox is the following.

- If two random subsets of the same size $l$ are independently chosen from a bigger set of size $N$, then how big should $l$ be to guarantee that the two subsets contain at least one common element?
  - Answer 1: if 100 % certainty is needed then it must be $l > N/2$.
  - Answer 2: if 99 % certainty is sufficient then it is enough that $l \approx \sqrt{N}$. 
Birthday paradox

One variant of the birthday paradox is the following.

- If two random subsets of the same size $l$ are independently chosen from a bigger set of size $N$, then how big should $l$ be to guarantee that the two subsets contain at least one common element?
- **Answer 1:** if 100% certainty is needed then it must be $l > N/2$.
- **Answer 2:** if 99% certainty is sufficient then it is enough that $l \approx \sqrt{N}$. 
One variant of the **birthday paradox** is the following.

- If two random subsets of the same size $l$ are independently chosen from a bigger set of size $N$, then how big should $l$ be to guarantee that the two subsets contain at least one common element?
- **Answer 1:** if 100 % certainty is needed then it must be $l > \frac{N}{2}$.
- **Answer 2:** if 99 % certainty is sufficient then it is enough that $l \approx \sqrt{N}$. 
Example

We have $l$ men and $l$ women in the same room. What is the probability that at least one man has the same birthday than one of the women?

- If $l < 183$ then the probability is less than 100 %.
  In fact, because some of the men can have the same birthday between themselves, (and similarly for women) there is no limit for $l$ above which the probability would raise to 100 %.

- If $l = 16$ then the probability is already 50 % (and raises quickly to 99 %).
We have \( l \) men and \( l \) women in the same room. What is the probability that at least one man has the same birthday than one of the women?

- If \( l < 183 \) then the probability is less than 100 \%. In fact, because some of the men can have the same birthday between themselves, (and similarly for women) there is no limit for \( l \) above which the probability would raise to 100 \%.
- If \( l = 16 \) then the probability is already 50 \% (and raises quickly to 99 \%).
Collision-resistance

- Assume hash length is $k = 128$.
  For example, generated by AES with block length of 128 bits.
- How to create collisions?
- Create $2^{64}$ versions of a document $m$ that is favorable to the signer. Compute hash for each message version.
- Repeat the same for an evil message $m'$.
- Birthday paradox guarantees that we have a good chance of finding two versions $(m, m')$ such that $h(m) = h(m')$.
- Learning: hash length must be at least 160 bits.
Collision-resistance

- Assume hash length is $k = 128$. For example, generated by AES with block length of 128 bits.
- How to create collisions?
  - Create $2^{64}$ versions of a document $m$ that is favorable to the signer. Compute hash for each message version.
  - Repeat the same for an evil message $m'$.
  - Birthday paradox guarantees that we have a good chance of finding two versions $(m, m')$ such that $h(m) = h(m')$.
  - Learning: hash length must be at least 160 bits.
Collision-resistance

- Assume hash length is $k = 128$. For example, generated by AES with block length of 128 bits.
- How to create collisions?
- Create $2^{64}$ versions of a document $m$ that is favorable to the signer. Compute hash for each message version.
- Repeat the same for an evil message $m'$.
- Birthday paradox guarantees that we have a good chance of finding two versions $(m, m')$ such that $h(m) = h(m')$.
- Learning: hash length must be at least 160 bits.
Assume hash length is $k = 128$. For example, generated by AES with block length of 128 bits.

How to create collisions?

Create $2^{64}$ versions of a document $m$ that is favorable to the signer. Compute hash for each message version.

Repeat the same for an evil message $m'$.

Birthday paradox guarantees that we have a good chance of finding two versions $(m, m')$ such that $h(m) = h(m')$.

Learning: hash length must be at least 160 bits.
Collision-resistance

- Assume hash length is $k = 128$.
  For example, generated by AES with block length of 128 bits.
- How to create collisions?
- Create $2^{64}$ versions of a document $m$ that is favorable to the signer.
  Compute hash for each message version.
- Repeat the same for an evil message $m'$.
- Birthday paradox guarantees that we have a good chance of finding two versions $(m, m')$ such that $h(m) = h(m')$.
- Learning: hash length must be at least 160 bits.
Collision-resistance

- Assume hash length is $k = 128$. For example, generated by AES with block length of 128 bits.
- How to create collisions?
- Create $2^{64}$ versions of a document $m$ that is favorable to the signer. Compute hash for each message version.
- Repeat the same for an evil message $m'$.
- Birthday paradox guarantees that we have a good chance of finding two versions $(m, m')$ such that $h(m) = h(m')$.
- Learning: hash length must be at least 160 bits.
Verification of the public keys

How to verify that a **public key** of Alice really belongs to Alice?

- **Solution:** certificates
- Certificate contains a public key digitally signed by a **trusted party**.
- Trusted party is called **Certificate Authority (CA)**.
How to verify that a **public key** of Alice really belongs to Alice?

- **Solution:** certificates
  - Certificate contains a public key digitally signed by a trusted party.
  - Trusted party is called **Certificate Authority (CA)**.
Verification of the public keys

How to verify that a public key of Alice really belongs to Alice?

- Solution: certificates
- Certificate contains a public key digitally signed by a trusted party.
- Trusted party is called Certificate Authority (CA).
Verification of the public keys

How to verify that a public key of Alice really belongs to Alice?

- Solution: certificates
- Certificate contains a public key digitally signed by a trusted party.
- Trusted party is called Certificate Authority (CA).
How to verify that the public key of CA really belongs to the CA?

- The CA has its own certificate, signed by another CA.
- We get certificate chains.
- The public key of the topmost CA must be verified somehow directly. This corresponds to a root certificate that is trusted without the help of any other certificate.
- The Public Key Infrastructure contains all CAs and some other elements that manage e.g. registration of public keys and revocation of certificates.
How to verify that the public key of CA really belongs to the CA?

- The CA has its own certificate, signed by another CA.
- We get certificate chains.
  - The public key of the topmost CA must be verified somehow directly. This corresponds to a root certificate that is trusted without the help of any other certificate.
- The Public Key Infrastructure contains all CAs and some other elements that manage e.g. registration of public keys and revocation of certificates.
How to verify that the public key of CA really belongs to the CA?

- The CA has its own certificate, signed by another CA.
- We get certificate chains.
- The public key of the topmost CA must be verified somehow directly. This corresponds to a root certificate that is trusted without the help of any other certificate.
- The Public Key Infrastructure contains all CAs and some other elements that manage e.g. registration of public keys and revocation of certificates.
How to verify that the public key of CA really belongs to the CA?

- The CA has its own certificate, signed by another CA.
- We get certificate chains.
- The public key of the topmost CA must be verified somehow directly. This corresponds to a root certificate that is trusted without the help of any other certificate.
- The Public Key Infrastructure contains all CAs and some other elements that manage e.g. registration of public keys and revocation of certificates.
PKI structure