Table of contents

Digital Signatures

Introduction and Definitions

A **digital signature** is a string of data associating a message with an originating entity, the digital equivalent of a handwritten signature.

Classification:

- **With appendix**: requires the original message for verification (ElGamal, DSS)
- **With message recovery**: allows reconstruction of the message (RSA, Rabin)

Signatures primarily use **asymmetric cryptography** to explicitly identify an entity.

arbitrated digital signatures: symmetric technology + Trusted Third Parties (TTP)

Original Text

Digital signature: string of data allowing to associate a message (in digital form) with an originating entity.
Digital signature scheme: generation algorithm + verification algorithm.
Signature process: message formatting + signature generation algorithm.
Verification process: verification algorithm + (message reconstruction).
Classification of digital signatures:

- **Digital signatures with appendix** which require the presence of the original message to verify the validity of the signature. These are the most commonly used. Examples: ElGamal, DSS.
- **Digital signatures with message recovery** which offer, in addition, the possibility

to reconstruct the message from the signature. Examples: RSA, Rabin.

Digital signatures are mostly based on **asymmetric crypto** because the concept of a shared key is not suitable for the need to identify an entity explicitly.
Commitments similar to those obtained with a public key signature (such as origin non-repudiation) can however be obtained with symmetric technology and trusted third parties (TTP). These methods are called: **arbitrated digital signatures**.

Quick Revision

Digital signature = string associating message + entity
Two types

- with appendix (requires original message)
- with recovery (reconstructs the message)

Based on asymmetric crypto

Formal Framework: Signatures with Appendix

Working spaces:

- M: message space
- M_h: hashed messages where $m_h = H(m)$ with H a hash function
- S: space of possible signatures

Operation:

Each entity A defines:

- $S_A : M_h \to S$ (signature function, uses private key)
- $V_A : M_h \times S \to \{\text{true}, \text{false}\}$ (verification, uses public key)

With $V_A(m_h, s) = \text{true}$ if and only if $S_A(m_h) = s$

Essential properties:

- S_A and V_A easy to compute with the correct keys
- Impossible to find a valid (m', s') without A's private key

Original Text

We assume that each entity has a private key to sign messages and an authentic copy of the public keys of correspondents.
Notation: - M: Message space - M_h: $m_h = H(m)$ with $m \in M$, $m_h \in M_h$ and H a hash function - S: Space of values that can be obtained by a signature process
Description:
Each entity defines an injective function $S_A : M_h \to S$ (i.e., the signature)
The function S_A gives rise to a function V_A: $V_A : M_h \times S \to \{\text{true}, \text{false}\}$ (i.e., the verification)
such that $\forall m_h \in M_h$, $s \in S$, we have: $V_A(m_h, s) = \text{true}$ if $S_A(m_h) = s$ and $V_A(m_h, s) =$ false otherwise
The operations S_A require the private key of A while the operations V_A use the public key of A.
Some simple properties:

- The operations S_A and V_A must be easy to compute (having the corresponding keys)
- It is impossible (computationally) for an entity not having the private key of A to find an m' and an s' with $m' \in M$ and $s' \in S$ such that $V_A(m'_h, s') = \text{true}$ with $m'_h = H(m')$.

Quick Revision

Signature: $S_A(m_h) = s$ (private key).
Verification: $V_A(m_h, s)$ (public key).
Impossible to forge without private key.

Formal Framework: Signatures with Recovery

Additional spaces:

- M_S: space on which the signature is applied
- $R : M \to M_S$: redundancy function (injective, invertible, public)
- $M_R = \text{Im}(R)$: image of R

Operation:

- Signature: $S_A : M_S \to S$ (injective)
- Verification: $V_A : S \to M_S$ with $V_A \circ S_A = \text{Identity}$

Generation:

1. Compute $m_R = R(m)$ and $s = S_A(m_R)$
2. Publish s as A's signature on m

Verification:

1. Compute $m_R = V_A(s)$ with public key
2. Verify $m_R \in M_R$ (reject otherwise)
3. Reconstruct $m = R^{-1}(m_R)$

Redundancy function:

Essential for security. If $M_R = M_S$, it becomes trivial to forge signatures.

Example: $R(m) = m \parallel m$ (concatenation). Forgery probability: $(1/2)^n$ for an n-bit message.

Original Text

Notation: in addition to the previous definitions, we have:
M_S: The space of elements on which a signature can be applied.
R: An injective function: $M \to M_S$, called the redundancy function. It must be invertible and public.
$M_R = \text{Im}(R)$
Description:
Each entity defines an injective function $S_A : M_S \to S$ (i.e., the signature)
The function S_A gives rise to a function $V_A : S \to M_S$ (i.e., the verification) such that $V_A \circ S_A =$ Identity on M_S
Note that verification is performed without the private key of A
Signature generation: 1. Compute $m_R = R(m)$ and $s = S_A(m_R)$ 2. Make s public as A's signature on m. This allows other entities to verify the signature and reconstruct m.
Verification: 1. Compute $m_R = V_A(s)$ (with A's public key) 2. Verify that $m_R \in M_R$ (otherwise reject the signature) 3. Reconstruct m by computing: $R^{-1}(m_R)$
Properties:

- The operations S_A and V_A must be easy to compute (having the corresponding keys)
- It is impossible (computationally) for an entity not having the private key of A to find an $s' \in S$ such that $V_A(s') \in M_R$

Remarks on the redundancy function:

- The choice of a redundancy function is **essential for the security** of the system.
- If $M_R = M_S$ and R and S_A are bijections respectively from M to M_R and from M_S to S, then M and S have identical size and, consequently, it is trivial to forge messages bearing A's signature.

Example of redundancy function: let $M = \{m : m \in \{0,1\}^n\}$ (n message size) and $M_S = \{t : t \in \{0,1\}^{2n}\}$. Let $R : M \to M_S$ such that $R(m) = m \parallel m$ ($\parallel$ being the concatenation of 2 messages). The probability of falling on such a message when trying to forge a message from a signature is: $|M_R|/|M_S| = (1/2)^n$, which is negligible for large messages.
Attention!: A redundancy function suitable for one digital signature scheme may cause vulnerabilities in another different one!

Quick Revision

With recovery: Redundancy function $R(m) = m_R$.
Signature $s = S_A(m_R)$.
Verification: $m_R = V_A(s)$, reconstruct $m = R^{-1}(m_R)$.
Redundancy crucial for security.

RSA Signature

Key generation:

- Choose two large prime numbers p and q
- Compute $n = pq$ and $\phi(n) = (p-1)(q-1)$
- Choose e with $\gcd(e, \phi(n)) = 1$
- Compute d such that $ed \equiv 1 \pmod{\phi(n)}$
- **Public key**: (n, e); **Private key**: d

Signature:

1. Compute $m_R = R(m)$ (redundancy function)
2. Compute $s = m_R^d \bmod n$
3. Send s

Verification:

1. Compute $m'_R = s^e \bmod n$
2. Verify $m'_R \in M_R$ (reject otherwise)
3. Reconstruct $m = R^{-1}(m'_R)$

Variant with appendix:

- **Signature**: $m_h = H(m)$, then $s = m_h^d \bmod n$
- **Verification**: $m'_h = s^e \bmod n$, accept if $m'_h = H(m)$

Characteristics:

- Signature slower than verification (d large, e small)
- Differentiate signature/encryption keys
- Vulnerable to the same attacks as RSA encryption

Original Text

Key generation:

- Each entity (A) creates a key pair (public and private) as follows:
- A chooses the size of the modulus n (e.g., size(n) = 1024 or size(n) = 2048).
- A generates two prime numbers p and q of large size ($n/2$).
- A computes $n := pq$ and $\phi(n) = (p-1)(q-1)$.
- A generates the verification exponent e, with $1 < e < \phi(n)$ such that $\gcd(e, \phi(n)) = 1$.
- A computes the signature exponent d, such that: $ed \equiv 1 \pmod{\phi(n)}$ using the extended Euclidean algorithm or fast exponentiation.
- The pair (n, e) is A's public key; d is A's private key.

Signature:

- A computes the redundancy function of the message m: $m_R := R(m)$.
- A computes the signature: $s := m_R^d \bmod n$ and sends s to B.

Verification:

- Entity B obtains (n, e), the authentic public key of A.
- B computes $m'_R = s^e \bmod n$, verifies $m'_R \in M_R$ and rejects the signature if $m'_R \notin M_R$
- B retrieves the correctly signed message by A by computing: $m = R^{-1}(m'_R)$.

Remarks:
The proof of operation is identical to that of the encryption process. The order of exponentiation has no influence since:

$$ed \equiv de \equiv 1 \pmod{\phi(n)}$$

The process can also be used to produce **signatures with appendix** with the following modifications:
Signature: - A uses a hash function H and computes $m_h := H(m)$. - A computes the signature of m_h: $s := m_h^d \bmod n$ and sends the pair (m, s) to B.
Verification: - B computes $m'_h = s^e \bmod n$ and $H(m)$ and verifies the equality $m'_h = H(m)$. - If the equality is verified, B accepts the signature s of A on the message M.
The signature computation is **slower** than verification because of the difference in size between the exponent d (size(d) $\approx$ size($\phi(n)$)) and e.
The risks and attacks mentioned in the encryption process also apply to the signature.
It is advisable to **differentiate the key pairs for encryption and signature** since they require distinct storage, backup and update policies.

Quick Revision

RSA signature: $s = m_R^d \bmod n$ (private).
Verify: $m'_R = s^e \bmod n$ (public).
With appendix: $s = H(m)^d \bmod n$.
Signature slow, verification fast.

Blind Signatures

Principle: A sends information to B for signature. From the response, A can compute B's signature on a different message that B has never seen.

Exploitation of the multiplicative property of RSA:

$$(m_1 m_2)^e \equiv m_1^e m_2^e \equiv c_1 c_2 \pmod n$$

Blinding functions:

Let k be an integer with $\gcd(n, k) = 1$:

- **Blinding**: $f(m) = m \cdot k^e \bmod n$
- **Unblinding**: $g(m) = k^{-1} \cdot m \bmod n$

Result: $g(S_B(f(m))) = g(S_B(mk^e \bmod n)) = g(m^d k \bmod n) = m^d \bmod n = S_B(m)$

Protocol:

1. A → B: $m' = f(m)$ (blinded message)
2. A ← B: $s' = S_B(m')$ (signature of blinded message)
3. A computes $g(s')$ and obtains $S_B(m)$ (signature of original message)

Applications: Anonymous electronic cash, electronic voting systems.

Original Text

Scheme invented by Chaum.
Idea: A sends information to B for signature. B returns to A the signed information. From this signature, A can compute B's signature on another message chosen beforehand by A. This allows A to have a signature of B on a message that B has never seen (hence the name blind signature…).
In fact it is a vulnerability based on the **multiplicative property of RSA**: $(m_1 m_2)^e \equiv m_1^e m_2^e \equiv c_1 c_2 \pmod n$ which was exploited to make a new signature scheme.
Algorithm: Let S_B be the RSA signature of B with (n, e) and d, respectively the public

and private keys of B. Let k be a fixed integer with $\gcd(n, k) = 1$:
$f : \mathbb{Z}_n \to \mathbb{Z}_n$ with $f(m) = m \cdot k^e \bmod n$ (blinding function)
$g : \mathbb{Z}_n \to \mathbb{Z}_n$ with $g(m) = k^{-1} \cdot m \bmod n$ (unblinding function)
which gives:

$$g(S_B(f(m))) = g(S_B(mk^e \bmod n)) = g(m^d k \bmod n) = m^d \bmod n = S_B(m)$$

Protocol:

- A $\to$ B: $m' = f(m)$
- A $\leftarrow$ B: $s' = S_B(m')$
- A computes $g(s')$ and obtains the desired signature using (*).

Quick Revision

Blind signature: Exploits RSA multiplicativity.
Blinding $f(m) = m \cdot k^e$
Unblinding $g(m) = k^{-1} \cdot m$.
B signs $f(m)$, A obtains $S_B(m)$ without B seeing m.

Rabin Signature

Key generation:

- Generate two large prime numbers p and q
- Compute $n = pq$
- **Public key**: n; **Private key**: (p, q)

Signature:

1. Compute $m_R = R(m)$ (redundancy function)
2. Compute $s = \sqrt{m_R} \bmod n$ (square root mod n)
3. Send s (one of the 4 square roots)

Verification:

1. Compute $m'_R = s^2 \bmod n$
2. Verify $m'_R \in M_R$ (reject otherwise)
3. Reconstruct $m = R^{-1}(m'_R)$

Characteristics:

- Based on the SQROOTP problem (square roots mod composite)
- **Provably secure**: equivalent to factorization
- Vulnerable to active chosen-ciphertext attacks

Original Text

Key generation:

- Each entity (A) creates a key pair (public and private) as follows:
- A generates two random prime numbers p and q of large size ($\text{len}(pq) \geq 1024$).
- A computes $n := pq$.
- The public key of A is n, the private key of A is (p, q).

Signature:

- A computes the redundancy function of the message m: $m_R := R(m)$.
- A uses its private key to compute the signature: $s := m_R^{1/2} \bmod n$ using efficient algorithms to compute square roots mod p and mod q.
- A sends s to B (s is one of the 4 obtained square roots).

Verification:

- Entity B obtains n, the authentic public key of A.
- B computes $m'_R = s^2 \bmod n$, verifies $m'_R \in M_R$ and rejects the signature if $m'_R \notin M_R$
- B retrieves the correctly signed message by A by computing: $m = R^{-1}(m'_R)$.

Remarks:
The Rabin procedure is based on the **impossibility of finding square roots modulo a composite of unknown factorization** (SQROOTP problem).
The main interest of this algorithm lies in the fact that it has been **proven to be equivalent to factorization** (SQROOTP FACTP). This algorithm therefore belongs to the **provably secure** category for any passive attack.
Active attacks can, in some cases, compromise the security of the algorithm. More precisely, if we mount the following chosen ciphertext attack (we ask A to decrypt a chosen ciphertext):

- The attacker M generates an m and sends to A the ciphertext $c = m^2 \bmod n$.
- A responds with a root m_x among the 4 possible m_1, m_2, m_3, m_4.
- If $m \not\equiv m_x \pmod{n}$ (probability 0.5), M repeats with a new m.
- Otherwise, A computes $\gcd(m - m_x, n)$ and thus obtains one of the two factors of n...

This attack could be avoided if the procedure required sufficient redundancy in the plaintexts allowing A to identify without ambiguity which of the possible solutions is the original plaintext. In this case, A would always respond with m and discard the other solutions that do not have the predefined level of redundancy.

Quick Revision

Rabin: $s = \sqrt{m_R} \bmod n$.
Verify: $m'_R = s^2 \bmod n$.
Provably secure (equivalent to factorization).
Vulnerable to active chosen-ciphertext attacks.

ElGamal Signature

Key generation:

- Generate prime p and generator $\alpha \in \mathbb{Z}_p^*$
- Generate random secret a, compute $y = \alpha^a \bmod p$
- **Public key**: (p, α, y); **Private key**: a

Signature:

1. Compute $m_h = H(m)$
2. Generate random k with $\gcd(k, p-1) = 1$
3. Compute $r = \alpha^k \bmod p$
4. Compute $s = k^{-1}(m_h - ar) \bmod (p-1)$
5. Signature: (r, s)

Verification:

1. Verify $1 \leq r \leq p-2$ (reject otherwise)
2. Compute $v_1 = y^r r^s \bmod p$
3. Compute $v_2 = \alpha^{H(m)} \bmod p$
4. Accept if $v_1 = v_2$

Proof: If $s \equiv k^{-1}(m_h - ar) \pmod{p-1}$, then $m_h \equiv ar + ks \pmod{p-1}$

So $v_2 = \alpha^{m_h} \equiv \alpha^{ar+ks} \equiv (\alpha^a)^r(\alpha^k)^s \equiv y^r r^s = v_1 \pmod{p}$

Characteristics:

- Works only with appendix (hash)
- Base of DSA (Digital Signature Algorithm)
- k must be unique for each signature

Original Text

Key generation:

- Each entity (A) creates a key pair (public and private) as follows:
- A generates a prime number p ($\text{len}(p) \geq 1024$ bits) and a generator α of $\mathbb{Z}_p^*$.
- A generates a random number a, such that $1 \leq a \leq p-2$ and computes $y := \alpha^a \bmod p$.
- The public key of A is (p, α, y), the private key of A is a.

Signature:

- A uses a hash function H and computes $m_h := H(m)$.
- A generates a random number k ($1 \leq k \leq p-2$ and $\gcd(k, p-1) = 1$) and computes $k^{-1} \bmod (p-1)$
- A computes $r := \alpha^k \bmod p$ and then $s := k^{-1}(m_h - ar) \bmod (p-1)$
- The signature of A on the message m is the pair (r, s).

Verification:

- Entity B obtains $(p, \alpha, \alpha^a \bmod p)$, the authentic public key of A.
- B verifies that $1 \leq r \leq p-2$, otherwise rejects the signature.
- B computes $v_1 := y^r r^s \bmod p$.
- B computes $H(m)$ and $v_2 := \alpha^{H(m)} \bmod p$
- B accepts the signature iff $v_1 = v_2$.

Remarks:
Proof that the scheme works: If $s \equiv k^{-1}(m_h - ar) \pmod{p-1}$, we have:

$$m_h \equiv (ar + ks) \pmod{p-1}$$

and

$$v_2 = \alpha^{H(m)} \bmod p$$

if, as we wish to show $m_h = H(m)$, by reducing exponents mod $(p-1)$, we can rewrite v_2:

$$v_2 \equiv \alpha^{ar+ks} \pmod{p}$$

On the other hand:

$$v_1 = y^r r^s \equiv \alpha^{ar}\alpha^{ks} \equiv \alpha^{ar+ks} \pmod{p}$$

Q.E.D.
By construction, the ElGamal scheme works **only with appendix** (result of applying a hash function). The Nyberg-Rueppel scheme introduces a variation allowing message recovery.
The **Digital Signature Algorithm (DSA)**, approved by the US National Institute of Standards and Technology has become the most commonly used signature standard. It is built on the basis of a direct derivative of the ElGamal scheme with the SHA-1 hash function.

Quick Revision

ElGamal: (r, s) with $r = \alpha^k \bmod p$, $s = k^{-1}(m_h - ar) \bmod (p-1)$.
Verify: $y^r r^s \overset{?}{=} \alpha^{H(m)} \bmod p$.
Base of DSA.
k unique crucial.

Signatures and Cryptocurrencies

Cryptocurrencies massively use digital signatures to authenticate transactions.

Bitcoin and Ethereum:

- Use **ECDSA** (Elliptic Curve Digital Signature Algorithm)
- Derivative of ElGamal on elliptic curves
- Security based on ECDLP

Transaction process:

Each spending/transmission requires:

- Signature with the **current holder's private key**
- The holder was the recipient of the previous transaction
- Each transaction forms an authentication chain

ECDSA advantages:

- Shorter keys for equivalent security
- More efficient computations
- Suitable for blockchain constraints

Original Text

Most cryptocurrencies are based on asymmetric cryptography. Bitcoin e.g. uses digital signatures to authenticate its transactions.
The spending or transmission of bitcoins requires the signature with the private key of the holder (who was in turn the recipient of the previous transaction).
Bitcoin and Ethereum use the **ECDSA (Elliptic Curve Digital Signature Algorithm)** algorithm derived from the ElGamal signature algorithm on elliptic curves whose security relies on ECDLP.
[Image: Diagram showing the Bitcoin transaction chain with signatures]
Source Image: Bitcoin: A Peer-to-Peer Electronic Cash System. Satoshi Nakamoto

Quick Revision

Cryptocurrencies: Bitcoin/Ethereum use **ECDSA** (ElGamal on elliptic curves).
Each transaction signed with holder's private key.
Security based on ECDLP.

Summary Table of Schemes

Class	Scheme	Message Recovery	Base Problem
Classical Signatures	RSA	Yes	RSAP
	Rabin	Yes	SQROOTP
	ElGamal	No	DLP
	DSS	No	DLP
One-time Signatures	Lamport	No	depends on OWF
	Bos-Chaum	No	depends on OWF
Undeniable Signatures	Chaum-van Antwerpen	No	DLP
Fail-Stop Signatures	van Heyst-Pedersen	No	DLP
Blind Signatures	Chaum	Yes	RSAP

Original Text

[Complete table with all above information]
The operation of One-time, Undeniable and Fail-Stop signature schemes can be consulted in [Men97].

Quick Revision

Classical signatures:

- RSA/Rabin (recovery)
- ElGamal/DSS (appendix)

Specialized:

- One-time
- Undeniable
- Fail-stop
- Blind

Base problems: RSAP, SQROOTP, DLP, depends on OWF.

Types of Attacks

Criteria for “breaking” a scheme:

- **Total Break**: Compute the private key or efficient generation algorithm
- **Selective forgery**: Generate signature for fixed message/class
- **Existential forgery**: Forge at least one signature (without message control)

Basic attacks:

- **Key-only**: Only the public key is known
- **Known-messages**: Access to signatures of known messages
- **Chosen-messages**: Attacker chooses messages to be signed
- **Adaptive chosen-messages**: Choice depends on previous responses

These attacks are equivalent to attacks on encryption systems (known/chosen-plaintext/ciphertext) but applied to messages.

Original Text

Criteria for “breaking” a digital signature scheme:

- **Total Break**: Compute the signer’s private key or an efficient (polynomial) algorithm to generate signatures.
- **Selective forgery**: The adversary is able to generate a valid signature for a fixed message (or class of messages).
- **Existential forgery**: The adversary is able to forge a signature for (at least) one message (which they do not control).

Basic attacks:

- **Key-only attacks**: The adversary only has knowledge of the signer’s public key.
- **Message-based attacks**: The adversary has access to signatures corresponding to:
  - known-messages
  - chosen-messages
  - adaptive chosen-messages

Equivalent to x-ciphertext attacks but with messages!

Quick Revision

Breaking signature:

- Total break (private key)
- selective forgery (fixed message)
- existential (one message)

Attacks:

- key-only
- known/chosen/adaptive-chosen-messages.