Digital Signatures

Federico Pintore [1]

[1] Mathematical Institute,
Oxford University

Outline

An Overview

- Digital signatures are used to provide integrity (or authenticity) in the public-key setting.
- They are the public-key analogue of MACs.
- For instance, software companies use them to allow the clients to verify that the software updates are authentic.
- Companies sign their updates using their secret keys. Then clients can verify the authenticity of the updates by verifying the validity of the signatures against the companies' public keys (are already known to the clients).

An Overview

- The owner of the public key acts as the sender, producing digital signatures using its private key SK.
- If a signature σ on a message m is verified correctly against a given public key PK, it ensures that the message was indeed sent by the owner of this public key (already known to potential verifiers) and the message was NOT modified in transit.
- In comparison to message authentication codes (MACs):
  - key distribution and management is hugely simplified;
  - signatures are publicly verifiable $\Rightarrow$ they are transferable (essential for certificates);
  - non-repudiation: signers cannot deny having signed a message;
  - MACs are shorter and more efficient to generate/verify.

Syntax

A digital signature scheme S consists of the following PPT algorithms:

- KeyGen(1^n): it takes the security parameter as input and returns a pair of keys (PK, SK), the public key PK and its matching secret key SK.
- Sign(SK, m): it takes a secret key SK, a message m from the message space $\mathcal{M}$ and returns a signature σ.
- Verify(PK, m, σ): it is a deterministic algorithm that takes a public key PK and a signature σ, and returns a bit, 1 for a valid signature and 0 for an invalid one.

Correctness: $\forall m \in \mathcal{M}$, and except with negligible probability over $(\text{PK}, \text{SK}) \leftarrow \text{KeyGen}(1^n)$, we have $\text{Verify}(\text{PK}, m, \text{Sign}(\text{SK}, m)) = 1$.

Security Definition: Unforgeability Experiment

We are assuming that parties are able to obtain a legitimate copy of the signer's public key.

Algorithm (Unforg$_{\mathcal{A},S}(n)$)

- KeyGen(1^n)*: it returns the keys* $(\mathsf{PK}, \mathsf{SK})$.
- *Signing Queries: the adversary* $\mathcal{A}$ *is given access to* PK *and to a signing oracle* Sign$(\mathsf{SK}, \cdot)$*. Let* Q *be the set of all messages that* $\mathcal{A}$ *asked its oracle to sign.*
- $\mathcal{A}$*'s output: a pair* (m^*, σ^*).
- ***If*** $Verify(\mathsf{PK}, m^*, \sigma^*) = 1$ *and* $m^* \notin Q$*, return 1,* ***else*** *return 0.*

Security Definition: Unforgeability Experiment

Definition

A signature scheme $S = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ is **existentially unforgeable under an adaptive chosen-message attack**, if for all PPT adversaries $\mathcal{A}$, we have

$$\Pr[\mathsf{Unforg}_{\mathcal{A},S}(n) = 1] \leq \mathsf{negl}(n)$$

Hash-and-Sign Paradigm

Let $S = (\text{KeyGen}, \text{Sign}, \text{Verify})$ be a digital signature scheme with message space $\{0,1\}^{\ell(n)}$, and let H be a hash function with output length $\ell(n)$.

We define a signature scheme $S' = (\text{KeyGen}', \text{Sign}', \text{Verify}')$ for arbitrary length messages as follows:

- $\text{KeyGen}'(1^n)$: it runs $\text{KeyGen}(1^n)$ to get a pair of keys (PK, SK), and $\text{KeyGen}_H(1^n)$ to get s. The public key is (PK, s), the secret key is (SK, s).
- $\text{Sign}'((\text{SK}, s), m \in \{0,1\}^*)$: it returns a signature $\sigma := \text{Sign}(\text{SK}, H^s(m))$.
- $\text{Verify}'((\text{PK}, s), m, \sigma)$: it that takes a public keys (s, PK), a message m and a signature σ, and ouptus 1 iff $\text{Verify}((\text{PK}, s), H^s(m), \sigma) = 1$.

Hash-and-Sign Paradigm

Theorem

If S is a secure digital signature scheme for messages of length $\ell(n)$ and H is a collision resistant hash function, then S' is a secure digital signature scheme for arbitrary-length messages.

Outline

RSA Signatures: Plain RSA

We define the plain RSA signature as follows:

- KeyGen(1^n): it runs a PPT algorithm GenRSA which, on input the security parameter n, returns a modulus $N = pq$, where p, q are two n-bit primes. It also outputs two integers e, d s.t. $e \cdot d = 1 \pmod{\phi(N)}$. PK $= (N, e)$ is the public key, SK $= (N, d)$ is the secret key.
- Sign(SK, m): it takes a secret key SK, a message $m \in \mathbb{Z}_N^*$ and returns a signature

$$\sigma := m^d$$

- Verify(PK, m, σ): it takes the public key PK, a message m and a signature σ, and returns a bit, 1 iff

$$m = \sigma^e$$

Security Analysis of Plain RSA

- The RSA assumption relative to GenRSA implies hardness of computing an eth root, i.e. of computing a signature for a uniform message m.
- What if the adversary can learn signatures on other messages?
- What about forging signatures on messages that the adversary can choose?
- **No message attack**: given a public key (N, e), pick $\sigma \in \mathbb{Z}_N^*$, compute the message as $m \leftarrow \sigma^e$ and output the forgery (m, σ).
- **Malleability**: knowing two valid signatures σ_1, σ_2 on two messages m_1, m_2, we can construct a valid signature on a new message $m = m_1 \cdot m_2$ as $\sigma \leftarrow \sigma_1 \cdot \sigma_2$.

RSA-Full Domain Hash (RSA-FDH)

The RSA-FDH signature scheme is defined as follows:

- KeyGen(1^n): it runs a PPT algorithm GenRSA which, on input the security parameter n, returns a modulus $N = pq$, where p, q are two n-bit primes, and two integers e, d s.t. $e \cdot d = 1 \pmod{\phi(N)}$. PK $= (N, e)$ is the public key, SK $= (N, d)$ is the secret key. A function $H : \{0,1\}^* \to \mathbb{Z}_N^*$ is also generated.
- Sign(SK, m): it takes a secret key SK, a message $m \in \{0,1\}^*$ and returns a signature

$$\sigma := H(m)^d$$

- Verify(PK, m, σ): it takes the public key PK, a message m and a signature σ, and returns a bit, 1 iff

$$H(m) = \sigma^e$$

Security of RSA-FDH

Theorem

If the RSA problem is hard relative to GenRSA *and* H *is modelled as a random oracle, then the digital signature RSA-FDH is secure.*

The RSA problem

Let GenRSA be a PPT algorithm that, on input n, outputs (N, p, q, e, d), where $N = pq$ - p, q are n-bit primes - and $[e]_{\varphi(N)}[d]_{\varphi(N)} = [1]_{\varphi(N)}$.

- In the experiment $\mathsf{RSA-inv}_{\mathcal{A},\mathsf{GenRSA}}(n)$, GenRSA is run on input n. The adversary is given N and e together with a uniform element $[y]_N \in \mathbb{Z}_N^*$. It has to determine $[x]_N \in \mathbb{Z}_N^*$ such that $([x]_N)^e = [y]_N$.
- The RSA problem is hard relative to GenModulus if, for all PPT adversaries $\mathcal{A}$, the success probability in the above experiment is negligible in n.
- The RSA assumption is the assumption that there exists a GenRSA relative to which the RSA problem is hard.

Security of (RSA-FDH)

Proof.

Let $\mathcal{A}$ be a PPT adversary against the $\text{Unforg}_{\mathcal{A},S}(n)$ experiment. We make the following assumptions during the security proof:

- If $\mathcal{A}$ queries the signing oracle for a signature on a message m, then it previously queried H for m.
- Same when $\mathcal{A}$ outputs a forgery (m, σ).
- $\mathcal{A}$ makes exactly $q(n)$ distinct queries to H.

We build an adversary $\mathcal{A}'$ against the $\text{RSA} - \text{inv}_{\mathcal{A},\text{GenRSA}}(n)$ experiment as follows: □

Security of RSA-FDH

Proof.

$\mathcal{A}'(N, e, y)$:

- It chooses uniform $j \in \{1, \cdots, q\}$
- It sends PK $= (N, e)$ to $\mathcal{A}$.
- It manages a storage table for triples of the form (m_i, σ_i, y_i), where $y_i = \sigma_i^e \mod N$ and $\mathcal{A}'$ has set $H(m_i) = y_i$.
- **Hash queries:** When $\mathcal{A}$ makes its i-th query, it answers as follows:
  - **if** $i = j$, the query is answered by y.
  - **else** a uniform $\sigma_i \leftarrow \mathbb{Z}_N^*$ is chosen and $y_i \leftarrow \sigma_i^e$ is computed. Then the tuple (m_i, σ_i, y_i) is stored in the table and y_i is returned (recall that f_e is a permutation of $\mathbb{Z}_N^*$).

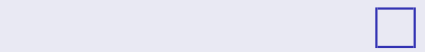
Security of (RSA-FDH)

Proof.

- **Signing queries:** when $\mathcal{A}$ makes its signing query on m, by hypothesis $m = m_i$, where m_i is already in the table, and $\mathcal{A}'$ answers as follows:
  - **if** $i = j$, then **aborts**.
  - **else** it finds the entry (m_i, σ_i, y_i) from the table, and returns σ_i to $\mathcal{A}$.
- $\mathcal{A}$ outputs its forgery (m, σ). If $m = m_j$ and $\sigma^e = y \mod N$, then $\mathcal{A}'$ outputs σ as an answer to the RSA experiment.

We observe that:

$$\Pr[\mathsf{RSA} - \mathsf{inv}_{\mathcal{A}',\mathsf{GenRSA}}(n) = 1] = \frac{\Pr[\mathsf{Unforg}_{\mathcal{A},S}(n)]}{q(n)}$$



More on RSA-FDH

- A signature scheme that can be viewed as a variant of RSA-FDH is included in the RSA PKCS #1 v2.1 standard.
- Some practical attacks on RSA-FDH are known if H has small output length (the range of H should be close to all $\mathbb{Z}_N^*$).
- Therefore, cryptographic hash functions such as SHA-1 are not suitable.

Outline

Schnorr Identification Scheme

- A three round interactive protocol that can be used to allow one party to authenticate itself.
- We will have two parties, a prover (e.g. a user) and a verifier (e.g a web server).
- The prover has a public key, together with the corresponding secret key. The verifier only knows the prover's public key.
- **Security of an identification scheme**: given an adversary that can eavesdrop on multiple executions of the protocol and who doesn't know the prover's secret key, it should NOT be able to fool the verifier into accepting.

Schnorr Identification Scheme

Let $\mathcal{G}$ be a PPT algorithm that, on input a security parameter n, outputs $(\mathbb{G}, q, q)$:

- $\mathbb{G}$ is a cyclic group $\mathbb{G}$;
- q is the order of $\mathbb{G}$, with $||q|| = n$;
- g is a generator of $\mathbb{G}$.

The prover runs $\mathcal{G}(n)$ and then generates its keys by choosing a uniform element $x \in \mathbb{Z}_q$: the public key PK is g^x, the secret key SK is x.

Schnorr Identification Scheme

Prover(x) **Verifier**$(\mathbb{G}, q, g, y = g^x)$

$k \leftarrow_{\$} \mathbb{Z}_q$

$I \leftarrow g^k$

I $\longrightarrow$

Challenge $c \leftarrow_{\$} \mathbb{Z}_q$

c $\longleftarrow$

$s \leftarrow cx + k \pmod{q}$

s $\longrightarrow$

$g^s \cdot y^{-c} \stackrel{?}{=} I$

Security of Schnorr Identification Scheme

Theorem

If the discrete logarithm problem is hard relative to $\mathcal{G}$, then the Schnorr identification is secure.

Sketch proof.

Does learning an *honest* transcript (I, c, s) help the adversary?

The adversary can **simulate** the transcript himself by reversing the order of the steps:

- It chooses uniform and independent $c^*, s^* \in \mathbb{Z}_q$
- It sets $I^* := g^{s^*} y^{-c^*}$.
- The transcript (I^*, c^*, s^*) is indistinguishable from an honest one (consider that f_{s^*} is a permutation of $\mathbb{Z}_N^*$, and the ElGamal's main idea).

Security of Schnorr Identification Scheme

Sketch proof.

Assuming that the adversary can, on input $y, I \in \mathbb{G}$, output a response s for any challenge c with high probability, then it can in particular respond with correct responses s_1, s_2 to two distinct challenge values $c_1, c_2 \in \mathbb{Z}_q$. But this means that we now have

$$g^{s_1} \cdot y^{-c_1} = I = g^{s_2} \cdot y^{-c_2}$$

Therefore, one can solve the discrete logarithm problem by finding

$$y = g^{\frac{s_2 - s_1}{c_2 - c_1}}.$$

Fiat-Shamir Transform

- It can be used to transform an interactive identification scheme into a (non-interactive) signature scheme.
- The signer runs the whole protocol by itself.
- It applies a hash function to (m, I) in order to generate the challenge c.
- The signature on m is now (c, s).
- The verifier can recompute I and check if the $c = H(I, m)$.

The Schnorr Signature Scheme

- KeyGen(1^n): it runs $\mathcal{G}$ on the security parameter n to obtain $(\mathbb{G}, q, g)$. It chooses a uniform $x \in \mathbb{Z}_q^*$ and sets $y = g^x$. The secret key is SK $= x$, whereas the public key is PK $= (\mathbb{G}, q, g, y)$. It also generates a hash function $H : \{0,1\}^* \to \mathbb{Z}_q$.
- Sign(SK, $m \in \{0,1\}^*$): on input a secret key SK and a message $m \in \mathbb{Z}_N^*$, it chooses a uniform $k \in \mathbb{Z}_q$, sets $I = g^k$ and computes $c := H(I, m)$ and $s := cx + k \mod q$. It finally returns a signature

$$\sigma := (c, s)$$

- Verify(PK, m, σ): it takes the public key PK, a message m and a signature σ, and computes $I := g^s \cdot y^{-c}$ and output 1 if

$$H(I, m) \stackrel{?}{=} c$$

Security of the Schnorr Signature Scheme

Theorem

If the discrete logarithm problem is hard relative to $\mathcal{G}$ and H is modelled as a random oracle, then the Schnorr signature scheme is secure.

Digital Signature Algorithm (DSA) and (ECDSA)

- Some of their versions go back to 1991.
- Both in the Digital Signature Standard (DSS) published by NIST.
- They are *based* on an identification protocol that is secure if discrete logarithm problem is hard.

DSA: underlying identification scheme

Prover(x) **Verifier**$(\mathbb{G}, q, g, y = g^x)$

$k \leftarrow_{\$} \mathbb{Z}_q^*$

$I \leftarrow g^k$

$$\xrightarrow{\quad I \quad}$$

Challenge $c, \alpha \leftarrow_{\$} \mathbb{Z}_q$

$$\xleftarrow{\quad c, \alpha \quad}$$

$s \leftarrow k^{-1} \cdot (\alpha + xc) \in \mathbb{Z}_q$

$$\xrightarrow{\quad s \quad}$$

$s \overset{?}{\neq} 0$

$g^{\alpha s^{-1}} \cdot y^{cs^{-1}} \overset{?}{=} I$

DSA: underlying identification scheme

- Correctness of the scheme: It is correct as long as $s \neq 0$, which only happens if $\alpha = -xc \mod q$. The probability that this happens is negligible.
- Security: based on the hardness of the discrete logarithm problem. Assume that the adversary $\mathcal{A}$, after it outputs an initial message I, can output correct responses (s_1, s_2) for two different challenges (c_1, α) and (c_2, α). An easy computation leads to $\log_g y$.

DSA

- KeyGen(1^n): it takes the security parameter n and runs $\mathcal{G}$ ot obtain $(\mathbb{G}, q, g)$. It chooses a uniform $x \in \mathbb{Z}_q^*$ and sets $y = g^x$. The secret key is SK $= x$, whereas the public key is PK $= (\mathbb{G}, q, g, y)$. It also generates two functions, $H : \{0,1\}^* \to \mathbb{Z}_q$ and $F : \mathbb{G} \to \mathbb{Z}_q$.
- Sign(SK, $m \in \{0,1\}^*$): given a secret key SK and a message $m \in \{0,1\}^*$, it chooses a uniform $k \in \mathbb{Z}_q^*$, sets $c := F(g^k)$ and computes $s := k^{-1} \cdot (H(m) + xc)$ in $\mathbb{Z}_q$. If $s = 0$ or $c = 0$, it starts again by choosing a fresh k. It finally returns the signature $\sigma := (c, s)$.
- Verify(PK, m, σ): it takes a public key PK, a message m and a signature $\sigma = (c, s)$ with $c, s \neq 0$. It outputs 1 if

$$g^{H(m) \cdot s^{-1}} \cdot y^{c \cdot s^{-1}} \stackrel{?}{=} c$$

Security of DSA/ECDSA

- Can be proven secure assuming the hardness of the discrete logarithm problem relative to $\mathcal{G}$ and modelling H and F as random oracles.
- Fine with H, but not with F. No known proofs for F as it is specified in the standard (F is a *simple* function, not intended to act as a random one).
- Choosing k properly is crucial. Knowing k, you can recover the secret key x.
- Using the same k leads to the private key as well!
- This is what the hackers did to recover the master secret key of Sony PS3 in 2010!

Bonus slide: Bilinear Maps (Pairings)

Let $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ be three groups of the same prime order p. A *pairing* is an efficiently computable function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, satisfying the following conditions:

1. $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$, for all $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$ and all $a, b \in \mathbb{Z}_p$.
2. *Non-degeneracy*: if g_1 is a generator of $\mathbb{G}_1$, g_2 is a generator of $\mathbb{G}_2$, then $e(g_1, g_2)$ is a generator of $\mathbb{G}_T$.

The Tate/Weil pairing maps any pair of elements from two groups of points on an elliptic curve, to an element in a subgroup of the multiplicative group of a finite field.

Pairing-based signatures

- Boneh-Lynn-Shacham signatures.
- Boneh-Boyen signatures.
- ...

Outline

Hash-based Signatures

- Signature schemes that are based on hash functions.
- No reliance on number-theoretic hardness assumptions.
- No reliance on random oracles.
- Believed to be post-quantum secure.

Lamport's Signature Scheme

- This was introduced by Leslie Lamport in 1979.
- It is a one-time secure signature scheme.
- By one-time we mean that the adversary $\mathcal{A}$ can only query the signing oracle on exactly one message during the $\text{Unforg}_{\mathcal{A},S}(n)$ experiment.
- One-time secure signature schemes are usually used to build other cryptosystems (including digital signatures) that achieve stronger notions of security.

Lamport's Signature Scheme

Example (Katz-Lindell book)

- Let's consider the example of a **3-bit** message.
- Let the private key and public keys be as follows:

$$\text{PK} = \begin{pmatrix} y_{1,0} & y_{2,0} & y_{3,0} \\ y_{1,1} & y_{2,1} & y_{3,1} \end{pmatrix} \quad \text{SK} = \begin{pmatrix} x_{1,0} & x_{2,0} & x_{3,0} \\ x_{1,1} & x_{2,1} & x_{3,1} \end{pmatrix}$$

- H is a cryptographic hash function, $\{x_{i,j}\}$ are chosen uniformly at random from $\{0,1\}^n$ and $y_{i,j} = H(x_{i,j})$, for $i = 1, 2, 3;\ \ j = 0, 1$.
- Given $m = \mathbf{011}$, the signature will be $\sigma = (x_{1,\mathbf{0}}, x_{2,\mathbf{1}}, x_{3,\mathbf{1}})$.
- Verification: given $m = \mathbf{011}$ and $\sigma = (x_{1,\mathbf{0}}, x_{2,\mathbf{1}}, x_{3,\mathbf{1}})$, Check if

$$H(x_{1,\mathbf{0}}) \stackrel{?}{=} y_{1,0} \quad H(x_{2,\mathbf{1}}) \stackrel{?}{=} y_{2,1} \quad H(x_{3,\mathbf{1}}) \stackrel{?}{=} y_{3,0}$$

Lamport's Signature Scheme

Given a function $H : \{0,1\}^* \to \{0,1\}^*$, the Lamport's signature scheme for messages of length $\ell(n)$ is defined as follows:

- KeyGen(1^n): it generates the following private and public keys

$$\mathsf{PK} = \begin{pmatrix} y_{1,0} & y_{2,0} & \cdots & y_{\ell,0} \\ y_{1,1} & y_{2,1} & \cdots & y_{\ell,1} \end{pmatrix} \quad \mathsf{SK} = \begin{pmatrix} x_{1,0} & x_{2,0} & \cdots & x_{\ell,0} \\ x_{1,1} & x_{2,1} & \cdots & x_{\ell,1} \end{pmatrix}$$

  where $\{x_{i,j}\}$ are chosen uniformly at random from $\{0,1\}^n$ and $y_{i,j} = H(x_{i,j})$, for $i = 1, \ldots, \ell; \quad j = 0, 1$.

- Sign$(\mathsf{SK}, m \in \{0,1\}^\ell)$: the signature will be $\sigma = (x_{1,\mathbf{m}_1}, \ldots, x_{\ell,\mathbf{m}_\ell})$, where $m = m_1 \ldots m_\ell$.

- Verify(PK, m, σ): on input $m = m_1 \ldots m_\ell$ and $\sigma = (\sigma_1, \ldots, \sigma_\ell)$, output 1 iff

$$H(\sigma_i) \stackrel{?}{=} y_{i,\mathbf{m}_i} \quad \forall i \in \{1, \ldots, \ell\}$$

Lamport's Signature Scheme

How can an attacker break the one-time security of the previous scheme?

Lamport's Signature Scheme

How can an attacker break the one-time security of the previous scheme?

- $\mathcal{A}$ can learn ONLY one signature on one message m of its choice.
- $\mathcal{A}$ has to output a signature on a **new** message m'.
- The new signature will then involve some **new** $x_{i,j}$, say $x_{1,1}$.
- If the forged signature is valid, $\mathcal{A}$ managed to compute the preimage of $y_{1,1}$, that is part of the public key.
- This cannot happen if H is a one-way function, i.e. finding the preimage is computationally difficult.

Security of Lamport's Signature Scheme

Theorem

If H is a one-way function, then the Lamport's signature scheme is a one-time secure signature scheme.

Outline

Certificates

- Used for the distribution of public keys.
- A trusted party is needed to start the process.
- A **digital certificate** is a digital signature that binds an entity to a certain public key.
- Assume that we have two parties, Alice with the pair of public/secret keys $(\mathrm{PK}_A, \mathrm{SK}_A)$, and Bob with $(\mathrm{PK}_B, \mathrm{SK}_B)$. We also assume that Alice knows PK_B.
- Alice can issue a **certificate** for Bob's key as follows:

$$\mathrm{cert}_{A \to B} := \mathrm{Sign}(\mathrm{SK}_A, \text{“Bob's key is } \mathrm{PK}_B\text{”})$$

- More identifying information about Bob are usually included in the certificate!

Certificates and Public Key Infrastructure (PKI)

- How do users learn PK_A? How can Alice be sure that Bob is the legitimate owner of PK_B? How do users decide whether to trust Alice? Such details determine a PKI.
- In the simplest form of PKI, we have a single **Certificate Authority (CA)**.
- A CA is a company (or a government agency) that certifies public keys.
- The mechanism by which a CA issues a certificate may vary from CA to CA.
- Everybody has to get a legitimate copy of PK_{CA}.
- The easiest way to distribute PK_{CA} is by physical means.
- Other ways include embedding CA's public key in the browser.
- Now if Charlie receives $\text{cert}_{CA \to B}$, he will be sure that the signed public key, i.e. PK_B, does indeed belong to Bob.

Certificates and Public Key Infrastructure (PKI)

- Delegations and certificate chains: a root CA can issue certificates to other CAs, say CA1, CA2, etc., that says: *"CA1's public key is* PK_{CA1} *and it is trusted to issue other certificates"*
- The "web of trust" model: a decentralised model - no reliance on a root CA. **Pretty Good Privacy (PGP)** is an example of this model. It is an email-encryption software for distribution of public keys.
- Invalidating Certificates:
  - Expiration: you can include an expiry date in the certificate.
  - Revocation: you can include a serial number in the certificate, while managing a **certificate revocation list (CRL)** that will be updated regularly.

Outline

SSL/TLS

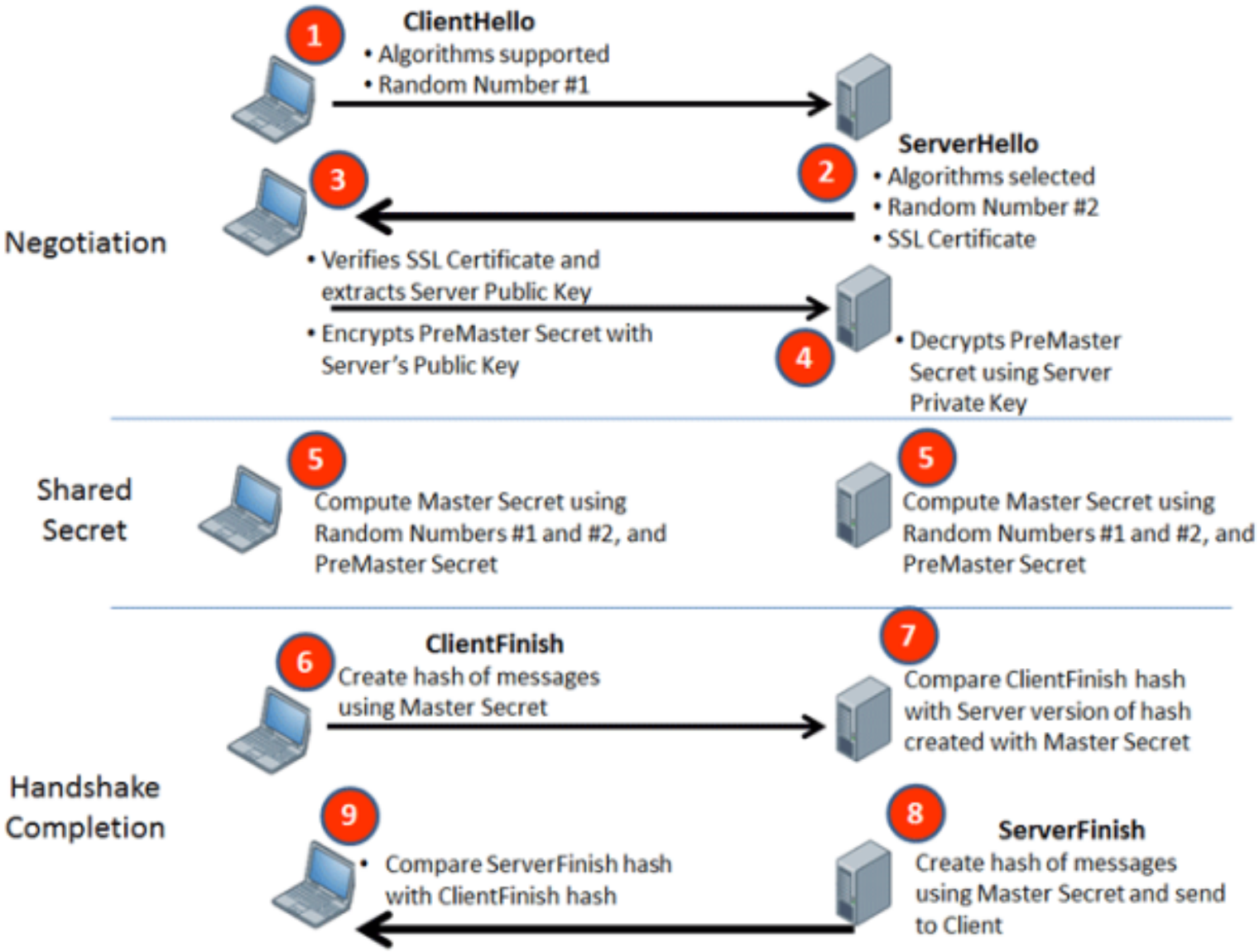
- The TLS protocol is used to secure communication over the web.
- Secure Socket Layer (SSL) is the old version, that was developed by Netscape around mid-1990s.
- Transport Layer Security (TLS) is the new version. Major websites now support the (TLS 1.2), although 50% of the websites still use TLS1.0!
- A given client (web browser) and a given server (website) can use TLS to agree on some shared keys which they will use to encrypt and authenticate their communication.

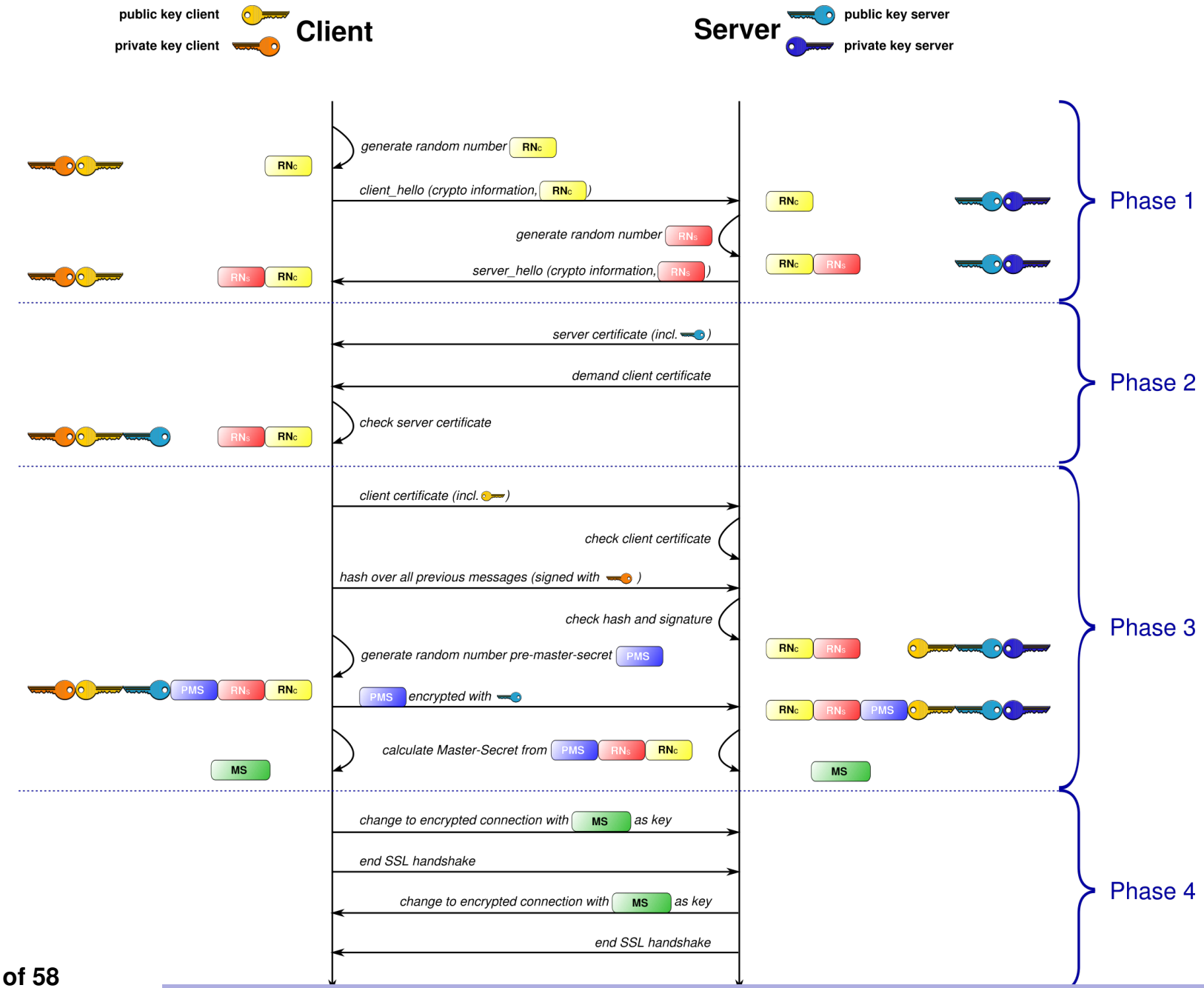
SSL/TLS

TLS consists of two phases:

- **The handshake protocol:** it performs an authenticated key exchange mechanism to establish the shared keys.
- **The record layer protocol:** it usually uses the shared keys to encrypt/authenticate the communication between parties. It first authenticates servers (that have certificates) to clients, and then clients can authenticate themselves to servers at the application level by using passwords.

(Next slide's diagrams from `http://archive.apachecon.com` and `https://www.identrustssl.com/learn.html`)


1
ClientHello
• Algorithms supported
• Random Number #1
2
ServerHello
• Algorithms selected
• Random Number #2
• SSL Certificate
3
Negotiation
• Verifies SSL Certificate and extracts Server Public Key
• Encrypts PreMaster Secret with Server's Public Key
4
• Decrypts PreMaster Secret using Server Private Key
5
Shared Secret
Compute Master Secret using Random Numbers #1 and #2, and PreMaster Secret
5
Compute Master Secret using Random Numbers #1 and #2, and PreMaster Secret
6
ClientFinish
Create hash of messages using Master Secret
7
Compare ClientFinish hash with Server version of hash created with Master Secret
Handshake Completion
9
• Compare ServerFinish hash with ClientFinish hash
8
ServerFinish
Create hash of messages using Master Secret and send to Client


public key client
private key client
Client
Server
public key server
private key server
generate random number RNc
client_hello (crypto information, RNc)
generate random number RNs
server_hello (crypto information, RNs)
Phase 1
server certificate (incl.)
demand client certificate
check server certificate
Phase 2
client certificate (incl.)
check client certificate
hash over all previous messages (signed with)
check hash and signature
generate random number pre-master-secret PMS
PMS encrypted with
calculate Master-Secret from PMS RNs RNc
MS
Phase 3
change to encrypted connection with MS as key
end SSL handshake
change to encrypted connection with MS as key
end SSL handshake
Phase 4

The Handshake Protocol - 1

- **Step 1:** $C \to S$: (ciphersuites, nonce N_C). % *A nonce is a uniform value*.
- **Step 2:** $S \to C$: (Latest version of TLS it supports, ciphersuites, PK_S, $\mathsf{cert}_{i \to S}$, nonce N_S).
- **Step 3:**
  - C verifies the certificate against the PK of CA_i, it makes sure it is not revoked or expired. If it is valid, it will use PK_S as the server's public key.
  - $(c, \mathsf{pmk}) \leftarrow \mathsf{Encaps}_{\mathsf{PK}_S}(1^n)$
  - $\mathsf{mk} \leftarrow$ **key-derivation function**(pmk, N_C, N_S)
  - $(k_C, k'_C, k_S, k'_S) \leftarrow \mathsf{PRG}(\mathsf{mk})$
  - $\tau_C \leftarrow \mathsf{MAC}_{\mathsf{mk}}(\text{transcript : all exchanged messages})$
  - $C \to S$: (c, τ_C)

The Handshake Protocol - 2

- **Step 4**:
  - S computes pmk $\leftarrow \text{Decaps}_{\text{SK}_S}(c)$.
  - mk $\leftarrow$ **key-derivation function**(pmk, N_C, N_S).
  - $(k_C, k'_C, k_S, k'_S) \leftarrow \text{PRG}(\text{mk})$.
  - **If** $\text{Verify}_{\text{mk}}(\text{transcript}, \tau_C) \neq 1$, then S aborts.
  - **Else** $\tau_S \leftarrow \text{MAC}_{\text{mk}}(\text{transcript}'\text{: transcript} \cup \text{last message from } C)$.
  - $S \rightarrow C$: τ_S
- **Step 5: If** $\text{Verify}_{\text{mk}}(\text{transcript}', \tau_S) \neq 1$, C aborts.

At the end of the handshake protocol, the client C and the server S share the following symmetric keys: k_C, k'_C, k_S, k'_S.

The Record-Layer Protocol

- C will use k_C to encrypt and k'_C to authenticate all messages that it will send to S.
- S will do the same with k_S and k'_S.
- They will use sequence numbers to prevent replay attacks.
- Note that even TLS 1.2 uses MAC-then-Encrypt approach, which is problematic. So no wonder why we have lots of attacks on TLS!

Further Reading (1)

- Carlisle Adams and Steve Lloyd.
  Understanding PKI: concepts, standards, and deployment considerations.
  Addison-Wesley Professional, 2003.
- Mihir Bellare and Phillip Rogaway.
  Random oracles are practical: A paradigm for designing efficient protocols.
  In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993.
- Dan Boneh, Ben Lynn, and Hovav Shacham.
  Short signatures from the weil pairing.
  Journal of cryptology, 17(4):297–319, 2004.

Further Reading (2)

- Tim Dierks.
  The transport layer security (TLS) protocol version 1.2.
  2008.
- Carl Ellison and Bruce Schneier.
  Ten risks of PKI: What you're not being told about public key infrastructure.
  Comput Secur J, 16(1):1–7, 2000.
- Amos Fiat and Adi Shamir.
  How to prove yourself: Practical solutions to identification and signature problems.
  In *Advances in Cryptology—CRYPTO'86*, pages 186–194. Springer, 1987.

Further Reading (3)

- Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh, and Vitaly Shmatikov.
  The most dangerous code in the world: validating SSL certificates in non-browser software.
  In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 38–49. ACM, 2012.
- Hugo Krawczyk.
  Cryptographic extraction and key derivation: The hkdf scheme.
  In *Annual Cryptology Conference*, pages 631–648. Springer, 2010.

Further Reading (4)

- Hugo Krawczyk, Kenneth G Paterson, and Hoeteck Wee.
  On the security of the TLS protocol: A systematic analysis.
  In *Advances in Cryptology–CRYPTO 2013*, pages 429–448. Springer, 2013.
- Leslie Lamport.
  Constructing digital signatures from a one-way function.
  Technical report, Technical Report CSL-98, SRI International Palo Alto, 1979.
- William Stallings.
  Network security essentials: applications and standards.
  Pearson Education India, 2007.