Diffie–Hellman key-exchange is an anonymous key-exchange protocol: neither side knows who they are talking to. It is vulnerable to a “man-in-the-middle” attack.

What we require: authenticated key-exchange (not anonymous) and relies on a root of trust (e.g., a certificate authority)

On the web, one of the parties will authenticate himself by presenting a certificate.

To build authenticated key-exchange, we require more ingredients — namely, an integrity mechanism [e.g., a way to bind a message to a sender — a “public-key MAC” or digital signature].

We will resist when discussing the TLS protocol.

### Digital signature scheme

Consists of three algorithms:

- **Setup** $(1^n) \rightarrow (vk, sk)$: Outputs a verification key $vk$ and a signing key $sk$
- **Sign** $(sk, m) \rightarrow \sigma$: Takes the signing key $sk$ and a message $m$ and outputs a signature $\sigma$
- **Verify** $(vk, m, \sigma) \rightarrow 0/1$: Takes the verification key $vk$, a message $m$, and a signature $\sigma$, and outputs a bit $0/1$

Two requirements:

- **Correctness**: For all messages $m \in M$, $(vk, sk) \leftarrow \text{KeyGen}(1^n)$, then $\Pr[\text{Verify}(vk, m, \text{Sign}(sk, m)) = 1] = 1$. [Honesty-generated signatures always verify]
- **Unforgeability**: Very similar to MAC security. For all efficient adversaries $A$, $\Pr[A_s] = \Pr[A_{\sigma}] = \text{negl}(n)$, where $W$ is the output of the following experiment:

  \[
  \begin{array}{cccc}
  \text{adversary} & \text{challenger} \\
  & \downarrow \\
  & (vk, sk) \leftarrow \text{KeyGen}(1^n) \\
  \uparrow & \sigma \leftarrow \text{Sign}(sk, m) \\
  \end{array}
  \]

  Let $m_1, \ldots, m_q$ be the signing queries the adversary submits to the challenger. Then, $W = 1$ if and only if:

  $\text{Verify}(vk, m, \sigma) = 1$ and $m \notin \{m_1, \ldots, m_q\}$

  Adversary cannot produce a valid signature on a new message.

Exact analog of a MAC (slightly weaker unforgeability: require adversary to not be able to forge signature on new message)

- MAC security required that no forgery is possible on any message [needed for authenticated encryption]
- digital signature algorithms
- elliptic-curve DSA
- standards (widely used on the web — e.g., TLS)

It is possible to build digital signatures from discrete log based assumptions (DSA, ECDSA)

But construction not intuitive until we see zero knowledge proofs

We will first construct from RSA (trivial permutations)
We will now introduce some facts on composite-order groups:

Let \( N = pq \) be a product of two primes \( p, q \). Then, \( \mathbb{Z}_N = \{0, 1, \ldots, N-1\} \) is the additive group of integers modulo \( N \). Let \( \mathbb{Z}_N^* \) be the set of integers that are invertible (under multiplication) modulo \( N \).

\[ x \in \mathbb{Z}_N^* \text{ if and only if } \gcd(x, N) = 1 \]

Since \( N = pq \) and \( p, q \) are prime, \( \gcd(x, N) = 1 \) unless \( x \) is a multiple of \( p \) or \( q \):

\[ \mathbb{Z}_N^* = N - p - q + 1 = pq - p - q + 1 = (p-1)(q-1) = \phi(N) \]

Recall Lagrange's Theorem:

for all \( x \in \mathbb{Z}_N^* : x^\phi(N) = 1 \pmod{N} \]

[called Euler's theorem, but special case of Lagrange's theorem]

- important: "ring of exponents" operate modulo \( \phi(N) = (p-1)(q-1) \)

Hard problems in composite-order groups:
- Factoring: given \( N = pq \) where \( p \) and \( q \) are sampled from a suitable distribution over primes, output \( p, q \)
- Computing cube roots: Sample random \( x \in \mathbb{Z}_N^* \). Given \( y = x^3 \pmod{N} \), compute \( x \pmod{N} \)

\[ \Rightarrow \text{This problem is easy in } \mathbb{Z}_N^* \text{ when } 3 \not\mid (p-1) \text{. Namely, compute } 1 \pmod{3}, \text{ and then compute } y^3 \pmod{3}. \]

\[ \Rightarrow \text{Why does this procedure not work in } \mathbb{Z}_N^* \text{? Above procedure relies on computing } 3 \pmod{(\mathbb{Z}_N^*)^3} = 3 \pmod{\phi(N)} \]

Hardness of computing cube roots is the basis of the RSA assumption:

RSA assumption: Take \( p, q \in \text{Primes}(1^\lambda) \), and set \( N = pq \). Then, for all efficient adversaries \( A, \)

\[ \Pr[x \in \mathbb{Z}_N^*: y \leftarrow A(N, x); y^3 = x] = \negl(\lambda) \]

[more generally, can replace 3 with any e when \( \gcd(e, \phi(N)) = 1 \)]

\[ \Rightarrow \text{Hardness of RSA relies on } \phi(N) \text{ being hard to compute, and thus on hardness of factoring} \]

(Reverse direction factoring \( \Rightarrow \text{RSA is not known} \))

Hardness of factoring / RSA assumption:
- Best attack based on general number field sieve (GNFS) - runs in time \( \tilde{O}(N^{1/3}) \)

(some algorithm used to break discrete log over \( \mathbb{Z}_p^* \))

- For 112-bit security, use RSA-2048 (\( N \) is product of two 1024-bit primes)
- For 128-bit security, use RSA-3072
- Both prime factors should have similar bit length (ECM algorithm factors in time that scales with smaller factor)
RSA problem gives an instantiation of more general notion called a trapdoor permutation:

\[ \text{Frsa} : \mathbb{Z}_N^* \rightarrow \mathbb{Z}_N^* \]

\[ \text{Frsa} (x) := x^e \pmod{N} \text{ where } \gcd(N, e) = 1 \]

Given \( \varphi(N) \), we can compute \( d = e^{-1} \pmod{\varphi(N)} \). Observe that given \( d \), we can invert \( \text{Frsa} \):

\[ \text{F}^{-1}(x) := x^d \pmod{N} \]

Then, for all \( x \in \mathbb{Z}_N^* \):

\[ \text{Frsa} (\text{Frsa} (x)) = (x^e)^d = x \pmod{\varphi(N)} = x^1 = x \pmod{N}. \]

**Trapdoor permutations**:

A trapdoor permutation (TDP) on a domain \( X \) consists of three algorithms:

- \( \text{Setup} (1^n) \rightarrow (\text{pp}, \text{td}) \): Outputs public parameters \( \text{pp} \) and a trapdoor \( \text{td} \).
- \( \text{F}(\text{pp}, x) \rightarrow y \): On input the public parameters \( \text{pp} \) and input \( x \), outputs \( y \in X \)
- \( \text{F}^{-1}(\text{td}, y) \rightarrow x \): On input the trapdoor \( \text{td} \) and input \( y \), output \( x \in X \)

**Requirements**:

- Correctness: for all \( \text{pp} \) output by \( \text{Setup} \):
  - \( \text{F}(\text{pp}, \cdot) \) implements a permutation on \( X \)
  - \( \text{F}^{-1} (\text{td}, \text{F}(\text{pp}, x)) = x \) for all \( x \in X \).
- Security: \( \text{F}(\text{pp}, \cdot) \) is a one-way function (to an adversary who does not see the trapdoor)

**Naive approach** (common “textbook” approach) to build signatures:

Let \( (F, F^{-1}) \) be a trapdoor permutation.

- Verification key will be \( \text{pp} \) ) to sign a message \( m \), compute \( \sigma = F^{-1}(\text{td}, m) \)
- Signing key will be \( \text{td} \) ) to verify a signature, check \( m \equiv F(\text{pp}, \sigma) \)

Correct because:

\[ F(\text{pp}, \sigma) = F(\text{pp}, F^{-1}(\text{td}, m)) = m \]

Secure because \( F^{-1} \) is hard to compute without trapdoor (signing key)

\[ \rightarrow \text{This is not true! Security of TDP just says that } F \text{ is one-way. One-wayness just says function is hard} \]

\[ \rightarrow \text{to invert on a random input. But in the case of signatures, the message is the input. This is not only} \]

\[ \rightarrow \text{not random, but in fact, adversarially chosen!} \]

\[ \rightarrow \text{Very easy to attack. Consider the } 0 \text{-query adversary:} \]

Given verification key \( vk = \text{pp} \), compute \( F(\text{pp}, \sigma) \) for any \( \sigma \in X \)

Output \( m = F(\text{pp}, \sigma) \) and \( \sigma \)

\[ \rightarrow \text{By construction, } \sigma \text{ is a valid signature on the message } m, \text{ and the adversary succeeds} \]

\[ \text{with advantage } 1. \]

**Textbook RSA signatures**:

[NEVER USE THIS!]

\[ \text{Setup}(1^n) \rightarrow (N, e, d) \text{ where } N = \text{pp} \text{ and } e \cdot d = 1 \pmod{\varphi(N)} \]

Output \( vk = (N, e) \) and \( sk = d \)

\[ \text{Sign}(sk, m) \rightarrow \sigma \equiv m^d \pmod{N} \]

\[ \text{Verify}(vk, m, \sigma) \rightarrow \text{Output } 1 \text{ if } \sigma^e \equiv m \pmod{N} \]

\[ \rightarrow \text{Looks tempting (and simple)… but totally broken!} \]
Signatures from trapdoor permutations (the full domain hash):

In order to appeal to security of TDP, we need that the argument to $F^{-1}(\text{id}, \cdot)$ to be random.

Idea: hash the message first and sign the hash value (often called "hash-and-sign")

⇒ Another benefit: Allows signing long messages (much longer than domain size of TDP)

FDH construction:
- $\text{Setup}(1^n)$: Sample $(pp, \text{id}) \leftarrow \text{Setup}(1^n)$ for the TDP and output $vk = pp, sk = \text{id}$
- $\text{Sign}(sk, m)$: Output $\sigma \leftarrow F^{-1}(sk, H(m))$
- $\text{Verify}(vk, m, \sigma)$: Output 1 if $F(pp, \sigma) = H(m)$ and 0 otherwise

Theorem. If $F$ is a trapdoor permutation and $H$ is modeled as a random oracle, then the full domain hash signature scheme defined above is secure.

Proof. Let $A$ be an adversary for the FDH signature. We use $A$ to build an adversary $B$ for the trapdoor permutation:

Claim. If $A$ succeeds with advantage $\epsilon$, then it must query $H$ on $m^*$ with probability $\epsilon = \frac{1}{|X|}$.

Proof. Suppose $A$ does not query $m^*$. Now, $(m^*, \sigma^*)$ is a valid forgery only if $F(pp, \sigma^*) = H(m^*)$.

However, if $A$ makes $\leq m^*$ queries, value of $H(m^*)$ uniform and independent of $F(pp, \sigma^*)$. Thus, $A$ succeeds with prob. $\frac{1}{|X|}$.

Key idea: If $A$ succeeds, it will invert the TDP at $H(m^*)$. [Algorithm B will program the challenge $y$ for $H(m^*)$].

But which query is $m^*$?

Without loss of generality, assume $A$ queries $H$ on message $m$ before making a signing query to $m$.

Suppose $A$ makes at most $Q$ queries to the random oracle. Algorithm B will guess which random oracle query is $m^*$.

1. Algorithm B samples $i^* \leftarrow [Q]$.
2. When $A$ makes a query to $H$ on input $m$:
   - Sample $x^* \leftarrow X$. Let $y^* := F(pp, x^*)$.
   - Set $H(x^*)$ to $y^*$ and remember the mapping $m \mapsto (x^*, y^*)$.

On query $i^*$ to $H$ for message $m^*$:
- Respond with challenge $y^*$.
- When $A$ makes a signing query for message $m$:
  - If $m = m^*$, then algorithm B aborts and outputs $\bot$. 

Otherwise, \( B \) looks up mapping \( m \mapsto (x, y) \) and replies with \( x \).

3. If \( B \) does not abort and \( A \) outputs \((m^*, o^*)\) where \( m^* = m^* \), \( B \) outputs \( o^* \). Otherwise, it outputs \( \bot \).

By construction, all queries to \( H \) are answered properly (since \( x \) is uniform and \( F(pp, \cdot) \) is a permutation).

If \( A \) does not make signing query on \( m^* \), then all signing queries answered perfectly.
- With probability \( e^{-\frac{1}{r+1}} \), algorithm \( A \) will query \( H \) on \( m^* \), not make a signing query on \( m^* \), and forge a signature on \( m^* \).
- With probability \( \frac{1}{Q} \), \( m^* = m^* \) in which case \( B \) perfectly simulates the signature security game.

Algorithm \( B \) succeeds with probability at least \( \frac{1}{Q} \left( e^{-\frac{1}{r+1}} \right) = \frac{e}{Q} - \text{negl}(n) \).

Some (partial) attacks can exploit very small public exponent \( e \).

Recap: RSA-FDH signatures:

- Setup \((\mathbb{Z}_n^*)^2\): Sample modulus \( N \), e, d such that \( ed = 1 \mod 4(N) \) — typically \( e = 3 \) or \( e = 65537 \)
  - Output \( vk = (N, e) \) and \( sk = (N, d) \)
- Sign \((sk, m)\): \( \sigma \leftarrow H(m)^d \) [Here, we are assuming that \( H \) maps into \( \mathbb{Z}_n^* \)]
- Verify \((vk, m, \sigma)\): Output 1 if \( H(m) = \sigma^e \) and 0 otherwise.

Standard: PKCS1 v1.5 (typically used for signing certificates)
- Standard cryptographic hash functions hash into a 256-bit space (e.g., SHA-256), but FDH requires full domain
- PKCS1 v1.5 is a way to pad hashed message before signing:

```
00 01 FF FF ... FF FF 00 DI H(m)
```

- Padding important to protect against chosen message attacks (e.g., preprocess to find messages \( m_1, m_2, m_3 \) where \( H(m_1) = H(m_2) \cdot H(m_3) \))
  - (but this is not a full-domain hash and cannot prove security under RSA — can make stronger assumption...)

With probability \( e^{-\frac{1}{3}} \),