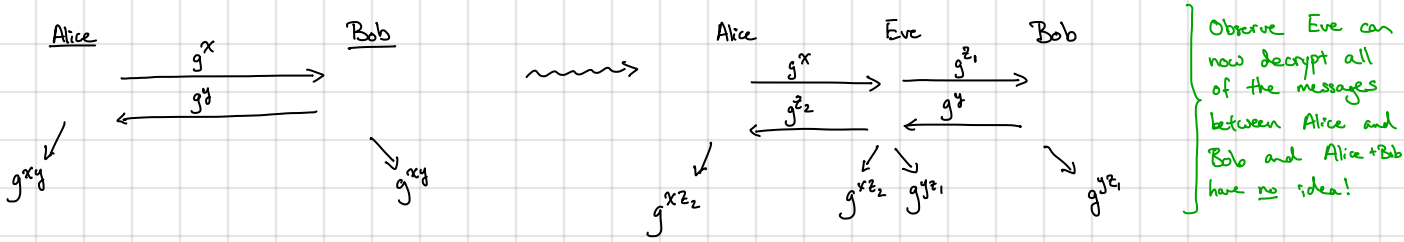
Diffie-Hellman key-exchange is an anonymous key-exchange protocol: neither side knows who they are talking to
↳ vulnerable to a "man-in-the-middle" attack

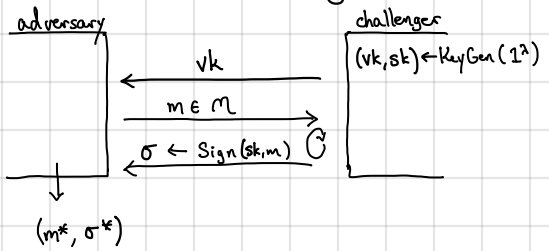


What we require: authenticated key-exchange (not anonymous) and relies on a root of trust (e.g., a certificate authority)
↳ On the web, one of the parties will authenticate themself by presenting a certificate

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 revisit when discussing the TLS protocol

Digital signature scheme: Consists of three algorithms:
- $\text{Setup}(1^\lambda) \to (vk, sk)$: Outputs a verification key vk and a signing key sk
- $\text{Sign}(sk, m) \to \sigma$: Takes the signing key sk and a message m and outputs a signature σ
- $\text{Verify}(vk, m, \sigma) \to 0/1$: Takes the verification key vk, a message m, and a signature σ, and outputs a bit $0/1$

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



Let $m_1, \ldots, m_Q$ be the signing queries the adversary submits to the challenger. Then, $W = 1$ if and only if:
$$\text{Verify}(vk, m^*, \sigma^*) = 1 \quad \text{and} \quad m^* \notin \{m_1, \ldots, m_Q\}$$
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 algorithm — 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 one-way functions
- In practice, digital signatures can also be built from hardness of factoring / RSA assumptions
↳ We will not cover these because these signature schemes are long and not post-quantum
↳ But still widely used in legacy systems

Lamport signatures: Let $f: X \to Y$ be a one-way function.

(n = length of message ($m = \{0,1\}^n$))

- $\text{Setup}(1^\lambda, 1^n)$: Sample $x_{i,b} \xleftarrow{R} X \ \forall i \in [n], b \in \{0,1\}$ and compute $y_{i,b} \leftarrow f(x_{i,b}) \ \forall i \in [2n], b \in \{0,1\}$

  Set

$$sk = \begin{array}{|c|c|c|c|} \hline x_{1,0} & x_{2,0} & \cdots & x_{n,0} \\ \hline x_{1,1} & x_{2,1} & \cdots & x_{n,1} \\ \hline \end{array} \qquad pk = \begin{array}{|c|c|c|c|} \hline y_{1,0} & y_{2,0} & \cdots & y_{n,0} \\ \hline y_{1,1} & y_{2,1} & \cdots & y_{n,1} \\ \hline \end{array}$$

- $\text{Sign}(sk, m \in \{0,1\}^n)$: Output $(x_{1,m_1}, \ldots, x_{n,m_n})$
- $\text{Verify}(vk, m, \sigma)$: Output 1 if $\forall i \in [n], f(x_{i,m_i}) = y_{i,m_i}$ and 0 otherwise.

Theorem. If f is one-way, then Lamport signatures are secure one-time signatures (i.e., where adversary can only make 1 signing query).

Proof. Suppose A is a one-time signature adversary. We construct B for f as follows:

1. Algorithm A receives challenge $y = f(x)$ where $x \xleftarrow{R} X$ from challenger.
2. Choose $i^* \xleftarrow{R} [n]$, $b^* \xleftarrow{R} \{0,1\}$ to program challenge. Sample $x_{i,b} \xleftarrow{R} X$, $y_{i,b} \leftarrow f(x_{i,b})$ for $(i,b) \neq (i^*, b^*)$.
   Set $pk = (y_{1,0}, y_{1,1}, \ldots, y_{n,0}, y_{n,1})$.
3. Send pk to B. If B makes signing query on $m = m_1, \ldots, m_n$:
   - If $m_{i^*} = b^*$, then abort.
   - Otherwise, reply with $(x_{1,m_1}, \ldots, x_{n,m_n})$.
4. After A outputs a forgery (m^*, σ^*), if $m^*_{i^*} \neq b^*$, then abort. Otherwise, output σ_{i^*}.

By construction, $m^*_{i^*} = b^*$ with probability $1/2n$. Thus if A succeed with probability ε, B succeeds with prob. $\varepsilon/2n$.

Limitations: One-time only [will fix later!] (two signatures allow recovering secret key!)

Long public keys, secret keys, and signatures
- Compose with CRHF to get $\text{poly}(\lambda)$-size parameters (independent of message length)
- Secret key can be derived from PRG (e.g., just λ bits)
- Public key can also be shortened to 2λ bits

Many combinatoric tricks to reduce signature size

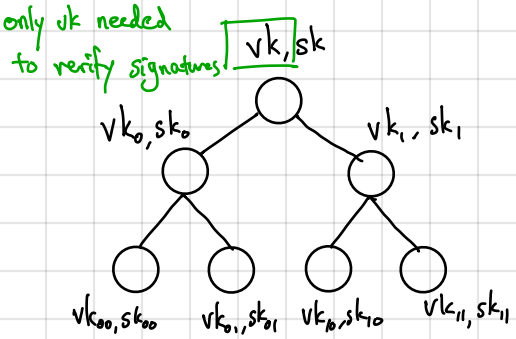
One-time signatures are very fast (only needs symmetric cryptography)
- Very useful in streaming setting: each packet in stream should be signed, but expensive to do so
  - Instead: include pk for one-time signature in first packet
    sign first packet using standard signature algorithm (public key)
    each packet includes OTS public key for next packet:

$$(m_0, vk_1), \sigma \to (m_1, vk_2), \sigma_1 \to (m_2, vk_3), \sigma_2, \ldots$$

σ: signed using many-time signature
σ_1: signed using secret key for vk_1
σ_2: signed using secret key for vk_2
} chaining signatures

Stateful many-time signatures from one-time signatures:

Idea: use a tree of one-time signatures:



- every node is associated with a key-pair for an OTS scheme
- each signing key used to sign verification keys of its children
- signing key for leaf nodes used to sign messages
- each leaf can only be used to sign one message — need to keep track of which nodes have been used (stateful signature)

Example: Signing message m using (vk_{00}, sk_{00}):

- $\sigma_0 \leftarrow \text{Sign}(sk, vk_0 \| vk_1)$
- $\sigma_{00} \leftarrow \text{Sign}(sk_0, vk_{00} \| vk_{01})$
- $\sigma_m \leftarrow \text{Sign}(sk_{00}, m)$
- Output $(vk_0 \| vk_1, vk_{00} \| vk_{01}, \sigma_0, \sigma_{00}, \sigma_m)$

To verify, check
- $\text{Verify}(vk, vk_0 \| vk_1, \sigma_0) = 1$
- $\text{Verify}(vk_0, vk_{00} \| vk_{01}, \sigma_{00}) = 1$
- $\text{Verify}(vk_{00}, m) = 1$

Only root vk needed here, all other keys included in σ

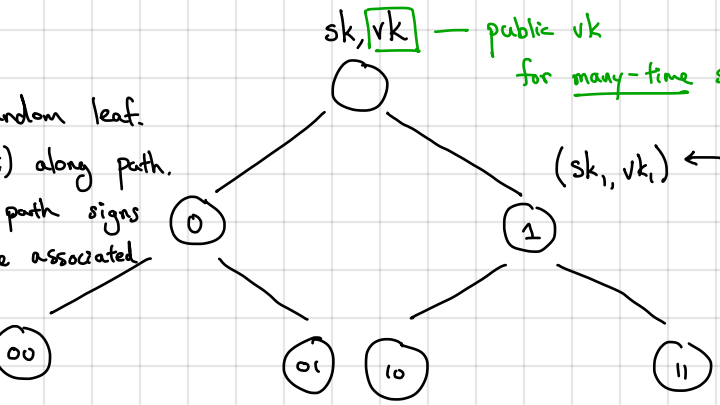
Security (Intuition):
- Keys for internal nodes only used to sign single message (verification keys of children)
- As long as leaf node never reused, then leaves are also only used once
- Security now reduces to one-time security of signature scheme

How to remove state?
- Consider a tree with 2^λ leaves and choose leaf at random for signing
- If we sign $\text{poly}(\lambda)$ messages, there will not be a collision in the leaf with $1 - \text{negl}(\lambda)$ probability
- Problem: Signing key is exponential (need to store $O(2^\lambda)$ signing keys)

Solution: Derive signing keys from a PRF!

$(vk_i, sk_i) \leftarrow \text{KeyGen}(1^\lambda ; \text{PRF}(k, i))$

(randomness to key-generation algorithm; i: node index; part of signing key)



$(sk_1, vk_1) \leftarrow \text{KeyGen}(1^\lambda ; \text{PRF}(k, 1))$

$(sk_{10}, vk_{10}) \leftarrow \text{KeyGen}(1^\lambda ; \text{PRF}(k, 10))$

To sign, choose random leaf.
Derive all (sk_i, vk_i) along path.
Each node along path signs verification node associated with children.
Leaf node signs message.
Signature contains complete validation path from root to leaf and signature of leaf on message.
Every internal node still signs only one message.