2.2 - MACs

Message Authentication Code

Efficiently computable function $M: \{0,1\}^{l} \times \{0,1\}^{*} \rightarrow \{0,1\}^{n}$, written $M(k, m) = t$

- k is the key; m is the message; t is the tag.
- Provides data integrity and data origin authentication
- No confidentiality or non-repudiation.

Security

- An adversary knows everything except the value of k.
- A MAC scheme is secure if it is existentially unforgeable against chosen-message attack.

Generic Attacks

- Guess the MAC of m
- Exhaustively search the keyspace.

MACs based on hash functions

- Secret prefix: $H(K \| m)$ - insecure
- Secret Suffix: $H(m \| k)$ - insecure
- Envelop: $H(k \| m \| k)$ - secure if MAC with m padded to a multiple of the block length of H

PKDF2

- Key derivation function
- Supposed to be slow
- Larger iteration => more security, slower performance

Pseudorandom generator

- A deterministic function PRF: $\underbrace{\{0,1\}^\lambda}_{\text{random seed}} \to \underbrace{\{0,1\}^\ell}_{\text{random-looking binary string}}$

Pseudorandom function

- A deterministic function PRF: $\underbrace{\{0,1\}^\lambda}_{\text{random seed}} \times \underbrace{\{0,1\}^*}_{\text{non-secret label}} \to \underbrace{\{0,1\}^\ell}_{\text{random-looking binary string.}}$

Key Derivation Function

- A deterministic function KDF: $\underbrace{\{0,1\}^\lambda}_{\text{random seed}} \times \underbrace{\{0,1\}^*}_{\text{non-secret label}} \to \underbrace{\{0,1\}^\ell}_{\text{random-looking binary string}}$

- Difference between KDF and PRF:

  ↳ KDF output should be indistinguishable from random even if the key k is non-random but has high entropy

Authenticated encryption

- Use separate keys for authentication and encryption

- Use separate keys for each party

- Create keys with KDFs

Encrypt-and-MAC (E&M)

Compute $c = Enc(m)$ and $t = MAC(m)$. Transmit $c \| t$.

Not secure. MAC does not ensure confidentiality.

MAC-then-encrypt (MtE)

Compute $t = MAC(m)$ and $c = Enc(m||t)$. Transmit c.

Not secure. SKES does not ensure integrity

Encrypt-then-MAC (EtM)

$c = Enc(m)$, $t = MAC(m)$, transmit $c||t$.

Secure if SKES and MAC are both secure.

AES-GCM

- Performs authentication and encryption
- Authentication is significantly faster than encryption
- Encryption and decryption can be parallelized.

2.3- Password Security.

Entropy

- Entropy measures the uncertainty in values generated from a random process
- If a password is chosen uniformly at random from a set of size 2^n, then its entropy is n bits, and requires around 2^{n-1} guesses on average to find it.
- Less uncertainty => Lower entropy, easier to guess.

Hashes for Login

- Advantages:
  - irreversible transformation to passwords.
  - almost no overhead for storage and login
- Disadvantages:
  - We cannot recover passwords
  - Attack creates a table of hashes to compare against database
  - Hashing is deterministic. => If passwords are the same then hashes are the same.
- Hash table: a table containing hashes of many/all possible passwords.
- Rainbow table: an example of a time-space tradeoff using hash chains.

  ↳ only works if the database stores the hash of the password H(password)

Salting

- Salting protects against rainbow tables
- Salting makes brute-force attack harder.

Password Hardening Function

- Computation of hash is not slowed down by a lot
- Brute-force attack slows down by a factor of $\underset{\text{\# iterations}}{\underline{10000}}$

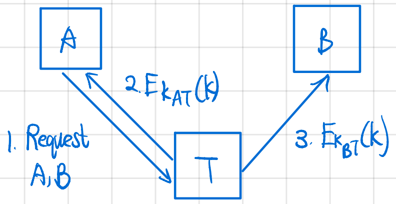
3.1 - Public Key Overview

Key Establishment

<u>Method 1:</u> Point-to-point key distribution

- This is generally not practical for large-scale applications.

<u>Method 2:</u> Use a Trusted Third Party (TTP) T



Drawbacks:

1. The TTP must be unconditionally trusted

2. The TTP is an attractive target

3. The TTP must be online.

Key Pair Generation

Each entity A generates (P_A, S_A)

- S_A is A's secret key

- P_A is A's public key.

It should be infeasible for an adversary to recover S_A from P_A.

Public Key Encryption

To encrypt a secret message m for Bob, Alice

1. Obtain an authentic copy of P_B.

2. Compute $c = \underbrace{E}_{\text{encryption}}(P_B, m)$

3. Send c to Bob.

To decrypt c, Bob computes $m = \underbrace{D}_{\text{decryption}}(S_B, c)$

Advantages of PKES

- No requirement for a secured channel
- Each user has only 1 key pair => better for key management.
- A signed message can be verified by anyone non-repudiation

Disadvantage of PKES

- PKES are much slower than SKES.

3.2 - RSA Encryption

RSA Encryption Scheme

Key generation:

1. Choose random primes p and q with $\log_2 p = \log_2 q = l/2$ usually $l = 2048$

2. Compute $n = pq$ and $\phi(n) = (p-1)(q-1)$

3. Choose an integer e with $1 < e < \phi(n)$, with $\gcd(e, \phi(n)) = 1$

4. Compute $d = e^{-1} \bmod \phi(n)$. The public key is (n, e) and private key is (n, d)

Message Space: $\mathcal{M} = \mathcal{C} = \mathbb{Z}_n^* = \{ m \in \mathbb{Z} : 0 \leq m < n \text{ and } \gcd(m, n) = 1 \}$

Encryption: $\mathcal{E}((n, e), m) = m^e \bmod n$

Decryption: $D((n, d), c) = c^d \bmod n$

Note: a/b is defined in $\mathbb{Z}_n$ if and only if $\gcd(b, n) = 1$.

Correctness of RSA

Let (n, e) be an RSA public key with private key (n, d). Then

$$D((n, d), \mathcal{E}((n, e), m)) = m$$

for all $m \in \mathbb{Z}_n$ such that $\gcd(m, n) = 1$.

Basic Modular Operations

Addition:	$O(l)$	
Subtraction:	$O(l)$	
Multiplication:	$O(l^2)$	
Inversion:	$O(l^3)$	
Exponentiation:	$O(l^3)$	* square-and-multiply, at most l squaring and at most l additions

3.3 Diffie-Hellman Key Exchange

Key Establishment Problem

Possible Solutions:

1. Use public-key cryptography which does not require shared secret keys

2. Use a key-exchange protocol, specifically designed to establish shared secrets from scratch.

Definition

The order of an element $x \in \mathbb{Z}_n^*$ is defined to be the smallest positive integer t such that $x^t = 1$ in $\mathbb{Z}_n^*$.

An element of $\mathbb{Z}_n^*$ is a generator if it has the maximum possible order.

Diffie-Hellman Key Exchange

A: Pick $a \in \mathbb{Z}_q$. Compute and send g^a. Receive g^b and compute $(g^b)^a = g^{ab}$

B: Pick $b \in \mathbb{Z}_q$, Compute and send g^b, Receive g^a and compute $(g^a)^b = g^{ab}$

The shared secret is g^{ab}

Diffie-Hellman vs. RSA

Diffie-Hellman

- Key exchange only: no arbitrary messages
- Interactive: must be online simultaneously
- Forward secrecy: cannot compromise past or future key exchanges even if one key exchange compromised.

RSA

- Public-key cryptosystem: can exchange any message chosen by the sender.
- Non-interactive: can decrypt encrypted message later
- No forward secrecy: a compromised private key compromises all past and future ciphertexts

Elgamal

Key generation:

- Choose $x \in_R \mathbb{Z}_q$, $pk = g^x \bmod p$ and $sk = x$

Encryption:

- Given $m \in \mathbb{Z}_p^*$, compute $\boxed{E(m) = (g^r, m \cdot \overset{\text{DH shared secret}}{g^{xr}}) \bmod p}$

Decryption:

- Given a ciphertext $(c_1, c_2) \in (\mathbb{Z}_p^*)^2$, compute $\boxed{D(c_1 \cdot c_2) = (c_1^{-1})^x \cdot c_2 \bmod p}$

Note: Elgamal is random.

3.4 - Public Key Encryption Security

Basic Assumption

Kerckhoff's Principle. Shannon's Maxim

- The adversary knows everything about the algorithm, except the private key k.

Adversary's Interaction

Passive attacks:

equivalent

- Key-only attack
- Chosen-plaintext attack
- Ciphertext-only attack

Active attacks:

- Chosen-ciphertext attack
- strongest → Adaptive chosen-ciphertext attack:
  - iteratively choose which ciphertexts to decrypt, based on the results of previous queries.

Adversary's Goal

Possible goals

- Total break: determine the private key (totally insecure)
- Decrypt a given ciphertext (one-way insecure)
- Learn some partial information (semantically insecure)

Security of RSA

RSA is totally insecure iff integer factorization is easy

RSA is one-way secure if the RSA problem is hard

RSA is not semantically secure under a ciphertext-only attack

- Let $c = m^e \bmod n$
- If $c=1$, then $m=1$
- If $c \neq 1$ then $m \neq 1$
- Why? Because RSA is deterministic and correct.

Semantic Security

A deterministic encryption algorithm cannot yield semantic security.

- Given a ciphertext $c = E_k(m)$
- Choose m' and compute $c' = E_k(m')$
- If $c' = c$ then $m' = m$, otherwise $m' \neq m$.

A randomized algorithm avoids this problem.

- Even with $c = E_{pk}(m)$ and $c' = E_{pk}(m)$, typically $c \neq c'$.

3.5 - Hybrid Encryption

Symmetric vs. Public

Symmetric:

- Fast
- Any bitstring of the right length is a valid key
- Any bitstring of the right length is a valid plaintext.
- Typical attack speed $\approx 2^{\ell}$ operations where ℓ is the key length.

Public

- Slow
- Keys have a special structure - not every bitstring of the right length is the key.
- Not every bitstring of the right length is a valid plaintext.
- Typical attack speed $\ll 2^{\ell}$ operations where ℓ is the key length

Hybrid Encryption

1. Use PKES to encrypt shared secret key.
2. Use SKES with the shared secret key to encrypt messages

Pros and Cons

Advantages

- Key management is the same as PKES
- Performance is close to SKES
- Security often improves

Disadvantages

- Attack surface increases

Basic Hybrid Encryption

- Let $(g, \mathcal{E}, D)$ be a PKES
- Let (E, D) be a SKES with ℓ-bit keys.
- Let (pk, sk) be a public/private key pair
- Let m be a message

Choose $k \in \{0,1\}^{\ell}$ at random, and compute and send (c_1, c_2):

$$\begin{cases} c_1 = \mathcal{E}(pk, k) & \text{encrypt symmetric key } k \text{ using } pk \\ c_2 = E(k, m) & \text{encrypt message using } k. \end{cases}$$

Improvement 1

Hash the key k before using it.

Encryption:

$$\begin{cases} c_1 = \mathcal{E}(pk, k) \\ c_2 = E(H(k), m)) \end{cases}$$

Decryption:

$$m = D(H(D(sk, c_1)), c_2)$$

Improvement 2

Example: Elgamal with a MAC

Encryption: choose r at random

$$(k_1, k_2) = H(g^{ar})$$
$$c = E(k_1, m)$$
$$t = MAC(k_2, c)$$

Send (g^r, c, t)

Decryption: Given $(\overset{g^r}{c_1}, \overset{c}{c_2}, \overset{t}{c_3})$

$$(\hat{k_1}, \hat{k_2}) = H(c_1{}^r)$$
$$\hat{t} = MAC(\hat{k_2}, c_2)$$
$$\hat{m} = D(\hat{k_1}, c_2)$$

Check $\hat{t} = c_3$? If true return $\hat{m}$, else reject

Diffie-Hellman Integrated Encryption Scheme (DHIES)

- DHIES is IND-CCA2, assuming
  - SKES is IND-CPA
  - MAC is secure (EUF-CMA)
  - H is a random oracle
  - DH problem is intractable

Improvement 3

Instead of a MAC, a simple hash check is enough

Encryption: For $m \in \{0,1\}^*$

$$\boxed{\begin{cases} c_1 = \mathcal{E}(pk, k) \\ c_2 = E(H_1(k), m) \\ c_3 = H_2(m, k) \end{cases}}$$

Decryption: Given (c_1, c_2, c_3)

$$k \leftarrow D(sk, c_1)$$
$$\hat{m} = D(H_1(k), c_2)$$

Check $H_2(\hat{m}, k) = c_3$? If true then return $\hat{m}$, else reject

Requirements:

- PKES is OW-CPA
- SKES is IND-CPA
- H_1 and H_2 are random oracles

3.6 - Elliptic Curve Cryptography

Elliptic curve cryptography

Use the points on an elliptic curve of the form $y^2 = x^3 + ax + b$ to create a group, then do Diffie-Hellman.

Group of points

For any elliptic curve $E: y^2 = x^3 + ax + b$, the set

identity element

$$\boxed{\{(x,y) : y^2 = x^3 + ax + b\} \cup \{O\}}$$

forms a group under the operation of point addition.

Point addition

Let P and Q be elements of elliptic curve group

- If $Q = O$, then $P + Q = P$
- If $P = O$, then $P + Q = Q$

$Q = -P$

- If $x_P = x_Q$ and $y_P = -y_Q$, then $P + Q = O$
- Otherwise use the formula

Elliptic curve Diffie-Hellman

- We write $\boxed{P^x = xP = \underbrace{P + P + \cdots + P + P}_{x \text{ occurrences of } P}}$ "scalar multiplication"
- A picks $x \in_R \mathbb{Z}_q$ and sends $xP \in E$ to B.
- B picks $y \in_R \mathbb{Z}_q$ and sends $yP \in E$ to A.
- Both compute $x(yP) = y(xP) = xyP \in E$.
- Use double-and-add (analog for square-and-multiply)

3.8- Digital Signature

RSA Signature

Key generation: $pk = (n, e)$, $sk = (n, d)$ like in RSA

Signature generation: To sign a message m,

1. Compute $s = m^d \bmod n$
2. The signature on m is s.

Signature verification: To verify s on m.

1. Obtain an authentic copy of the public key (n, e)
2. Compute $s^e \bmod n$
3. Accept iff $s^e \bmod n = m$.

Correctness Requirement

For a given key pair (pk, sk) produced by G.

$$\boxed{Ver(pk, m, Sign(sk, m)) = true}$$

for all $m \in M$.

Security - Adversary's Goals

strong

1. Total break: Recover the private key, or systematically forge signatures
2. Selective forgery: Given a message or a subset of messages, forge a signature for these messages.
3. Existential forgery: Forge a signature for some message.

weak

Attack Model

1. Key-only attack: The public key is known
2. Known-message attack: Some messages and their valid signature are known
3. Chosen-message attack: May choose some messages and obtain their signature ← strongest

Malleability of Basic RSA Function

Given $c = m^e \bmod n$, for any $x \in \mathbb{Z}_n^*$, we can construct c' encrypting mx by

$$c' = x^e \cdot c \bmod n$$

Digital Signature Summary

- Public key primitive providing data integrity, data origin authentication, and non-repudiation.
- Security goal: existential unforgeability against chosen-message attacks.

4.1 - Key Management

Public Key Distribution Problem

- Man-in-the-middle who replaces public keys can decrypt.

Public Key Distribution

- directly from subject.
- from a friend / friend of a friend ("web of trust")
- from a public directory (PGG key server, "public key infrastructure")

Web of Trust

Advantages
- simple
- free
- works well for a small number of users

Disadvantages
- relies on human judgement
- doesn't scale to large number of parties
- not appropriate for trust sensitive areas

Certificates and certificate authorities.

- Relies on trusted authorities (called certificate authorities) to vouch that public keys belong to certain subjects.
- Certificate: an assertion by a 3rd party that a particular key belongs to a particular entity.
- A digital certificate contains:
  - subject identity
  - subject's public key
  - validity period
  - the issuer's digital signature.

Certificate generation:

1. Obtain subject's public key
2. Verifying that the subject's identity.
3. Signing (using the CA's private key) the subject's public key and name.

Certificate revocation mechanisms

Certificate Revocation Lists (CRLs)

- Each CA can publish a file containing a list of certificates that have been revoked
- CRL address often included in certificate.

Online Certificate Status Protocol.

- An online service run by a CA to check in real-time if a certificate has been revoked.
- Not widely implemented
- Compromises user privacy

Public Key Infrastructure

- A set of systems for managing digital certificates.

Obtaining Public Key

Alice needs Bob's public key

1. Alice obtains Cert_{Bob}
2. Alice checks that the identity in Cert_{Bob}
3. Alice verifies CA's signature on Cert_{Bob} using CA's public key.

It provides confidentiality and integrity if

- CA checks the identity before issuing.
- CA does not issue fraudulent certificates
- Alice is certain of the CA's public key.

4.2 TLS and SSH

TLS

Transport Layer Security is a cryptographic tool that operates above the transport layer to provide security services to applications.

TLS Security Goals

- Provides authentication based on public key certificates.
  - server-to-client (always)
  - client-to-server (optional)

- Provides confidentiality and integrity of message transmission.

TLS Handshake Protocol

- Authentication: ensures that the connection really is with the server.
  - typically uses X.509 certificates

TLS Key Exchange

1. RSA

   - no forward security

   - not permitted in TLS 1.3

2. Ephemeral Diffie-Hellman.

   - has forward security

   - only permitted method in TLS 1.3.

TLS Security

TLS provides

- server-to-client authentication
- client-to-server authentication (optional)
- confidential communication with integrity and replay protection

TLS doesn't provide

- hide source/destination
- hide length information
- password-based authentication
- stop denial of service attacks.

Forward Secrecy

- An adversary who later learns the server's long-term private key is not able to read previous transmissions.
- Signed DH key exchange provides forward secrecy.

SSH protocol

- Provides public key authentication of server to clients and encrypted communications
- Runs over TCP.

SSH Security Goals

- Message Confidentiality - achieved using encryption
- Message Integrity - achieved using MAC.
- Message Replay Protection - achieved using counters and integrity protection
- Peer Authentication: server-to-client auth, client-to-server auth

Server authentication in SSH

- Based on public key digital signatures.
- Unlike TLS, (typically) does not use certificates, just a raw public key (hashed)

4.3 - Signal

Signal Goals

1. Long-lived sessions. The session lasts until events such as app reinstall or device change

2. Asynchronous setting. We can send message even if one party is offline.

3. Fresh session keys. Each message is encrypted / authenticated with a fresh session key.

4. Immediate decryption.

5. End-to-end encryption

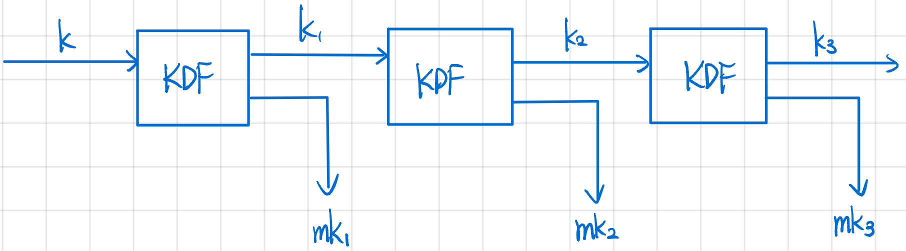
6. Forward secrecy.

7. Post-compromise security Parties recover from a state compromise

Forward Secrecy

"symmetric rachet"

Suppose Alice and Bob share a secret key k.



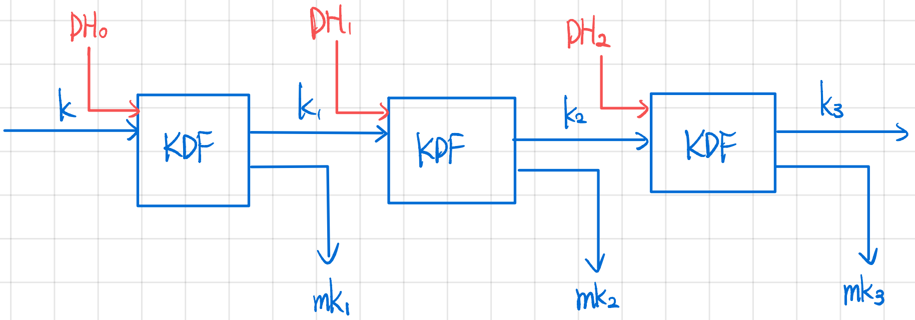
- Keys are deleted as soon as they are no longer needed.

- Given k_i and mk_i, an adversary can compute $k_{i+1}, mk_{i+1}, k_{i+2}, mk_{i+2}, \dots,$ but not $k_{i-1}, mk_{i-1}, \dots$.

Post Compromise Secrecy

"asymmetric rachet"

- Suppose Alice and Bob share a secret key k.
- A fresh ECDH is used each time the KDF is applied.



$DH_i = ECDH(X_i, Y_i)$

- Given k_i and mk_i, an adversary cannot compute k_{i-1}, mk_{i-1}, k_{i-2}, mk_{i-2}, ..., nor k_{i+1}, mk_{i+1}, k_{i+2}, mk_{i+2}, ...

Message Transmission

Each party maintains 3 key chains:

1. A root key chain
2. A sending key chain
3. A receiving key chain.

4.4 - Bitcoin.

Security properties

For the payer:

- Payer anonimity during payment
- Payer untracability. Others cannot tell whose coins are used in a particular payment

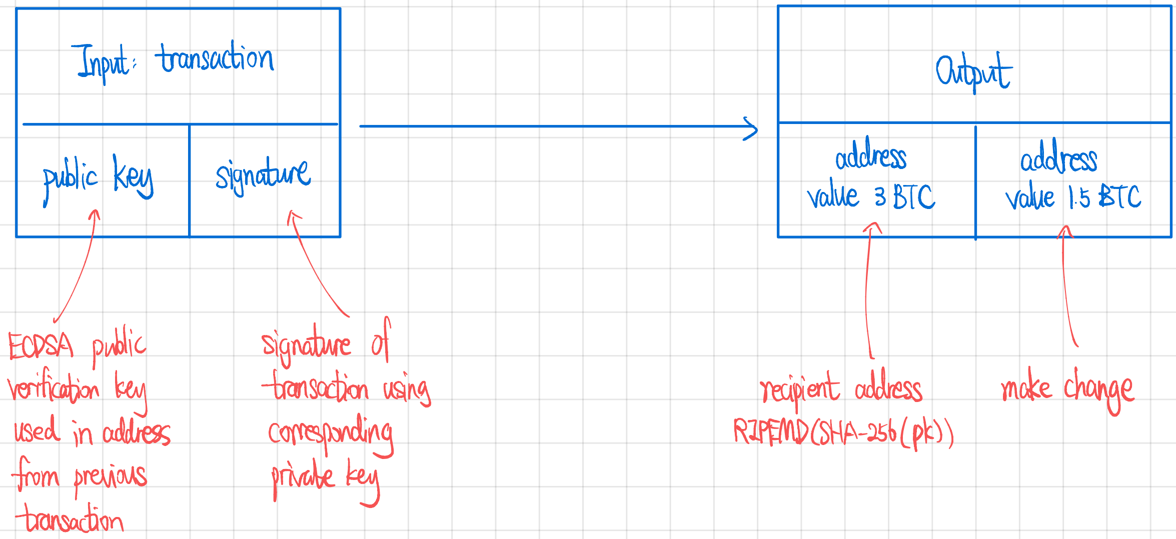
For the payee:

- Unforgeable coins. Forging valid looking coins should be infeasible
- No double-spending. A coin cannot be used more than once.

Basic Ideas

- Use public key for names
- Use transaction references for accounts
- Use digital signature to demonstrate ownership of currency.
- Distributed ledger: incentivize community to maintain

Transaction



Block and BlockChain

Block: header + a list of transactions

Blockchain: a sequence of blocks = a ledger of transactions

- Blockchains form a tree: Only the longest chain is considered to be valid by the community.
- Motivation: Whoever constructs the block includes one transaction paying themselves 6.25 BTC. "mining"
- Everyone is motivated on a single public ledger.
- The miners are trying to construct a block header where

$$\boxed{H(H(\text{block header} \,\|\, \text{solution})) \leqslant \text{difficulty target}}$$

Cryptographic ingredients

- Hash functions (SHA-256, RIPEMD-160)
- Cryptographic puzzles (Hashcash with SHA-256)
- ECDSA

4.5 Zero-Proof Knowledge

Basic Idea

A wants to prove to B that A knows something, without disclosing any information to B.

Commit - Challenge - Response

1. A generates a commitment and sends it to B
2. B generates a challenge and sends it to A
3. A generates a response and sends it to B
4. B verifies the response.

Zero knowledge

- B "learns nothing" if B could have generated all of the values he received on his own.
- i.e., there exist a simulator that outputs transcripts that are indistinguishable from real transcripts.
- B has to generate them in a different order.
- Honest Execution.
  1. Generate commitment
  2. Receive challenge
  3. Generate response
- Simulator
  1. Pick challenge
  2. Generate response
  3. Retroactively compute commitment.
- Order matters. A has to make a commitment H that will work for any challenge.
- The simulator can retroactively build the commitment to work for one particular challenge

Non-interactive proofs

If prover can pick commitment after challenge, then it's possible to fool the verifier.

Idea: challenge = hash of commitment

- secure assuming the hash is a random function.