Alice wants to access some resources controlled by Bob.

Bob is willing to provide them to Alice, but not to everyone.

Alice has to convince Bob that she really is Alice.

How?

This is the identification problem.

Passwords — the simplest scheme.

Alice and Bob have agreed on a common bit-string M.

Alice sends M to Bob. Bob verifies that it really received M and grants access to Alice.

Problems:

- An eavesdropper may learn M and impersonate Alice afterwards.
- Bob has to store M somewhere. If Bob's computer gets compromised then M has leaked.
- Alice may not use M to identify herself to Charlie.
  - Because Bob could impersonate her.
  - And if Bob's computer is compromised, then the attacker can impersonate her also to Charlie.
- If M is human-memorable then it typically has low entropy.

To prevent the leakage of M when Bob's computer is compromised, Bob only stores $h(M)$, where h is a one-way function.

The low entropy of M still allows it to be brute-forced.

If Bob has a database of $h(M)$-s for many different users then compromising him is especially attractive.

To reduce attractiveness, Bob stores not $h(M)$, but $(R, h(R \mathbin{||} M))$ for a random string R.

- different R-s for different users.

If M has high entropy, it cannot be memorable to humans.

It may be stored on a smart-card instead.

This smart-card may require a PIN to activate. It may lock after a couple of false PINs.

If Alice always sends the same M to Bob then the eavesdropper can impersonate her.

Use one-time passwords or randomization.

It is possible to use very little storage for a lot of passwords, if they are generated in a special way.

Let r be a random string. Let n be the number of passwords. Define

$$M_n = r$$
$$M_i = h(M_{i+1}) \quad (0 \leqslant i \leqslant n-1)$$

for a one-way hash function h.

Alice stores r. Bob stores M_0.

To identify itself for the i-th time, Alice recomputes M_i from r and sends it to Bob. Bob verifies that $h(M_i) = M_{i-1}$. Bob then stores M_i and may delete M_{i-1}.

If Alice always sends the same M to Bob then the eavesdropper can impersonate her.

Use one-time passwords or randomization.

The randomness has to come from Bob's side.

Challenge-response protocol:

- Bob generates a challenge x and sends it to Alice.
- Alice responds by computing something from x and M, and sending it to Bob.
- Bob verifies that the message sent by Alice was really computed from x and M in the prescribed way.

For example, let E be the encryption function of some symmetric cryptosystem.

- Bob generates a bit-string x and sends it to Alice. Also computes $y_B = E_M(x)$.
- Alice computes $y_A = E_M(x)$ and sends it to Bob.
- Bob verifies that $y_A = y_B$.

Problems:

- Alice cannot use M to identify herself to Charlie.
- The attacker impersonating Bob can mount a chosen-plaintext attack against M.

Let E and D be the encryption and decryption functions of some asymmetric cryptosystem.

Let M_s be Alice's secret key and M_p Alice's public key.

- Bob generates a bit-string x, computes $y = E_{M_p}(x)$ and sends it to Alice.

- Alice computes $x' = D_{M_s}(y)$ and sends it to Bob.

- Bob verifies that $x = x'$.

Good things:

- Alice never reveals M_s. She merely proves her knowledge of M_s.

- Hence Alice can use M_s to identify herself to Charlie.

But chosen-ciphertext attacks are possible.

Schnorr's identification scheme. Let η be the security parameter. Let p be a 1024-bit prime number, such that discrete log. in $\mathbb{Z}_p^*$ is hard and $q \mid (p-1)$ is a 160-bit prime number, such that $q > 2^\eta$. Let $g \in \mathbb{Z}_p^*$ have the order q.

Alice's secret key: $a \in \mathbb{Z}_q$. Public key: $h = g^a$.

Commitment Alice picks random $r \in \mathbb{Z}_q$, sends $x = g^r$ to Bob.

Challenge Bob chooses a random $k \in \{1, \ldots, 2^\eta\}$, sends it to Alice.

Response Alice sends $y = (r + ak) \bmod q$ to Bob.

Verification Bob accepts if $x = g^y / h^k$ in $\mathbb{Z}_p^*$.

This scheme is fast as commitments can be precomputed.

If Alice and Bob are honest then verification succeeds:

$$g^y/h^k = g^{(r+ak) \bmod q} \cdot (g^a)^{-k} = g^{r+ak} \cdot g^{-ak} = g^r = x \ .$$

Exercise. Suppose that Eve is able to predict Bob's choice of k at the challenge step. How can she then impersonate Alice?

Exercise. What happens if Alice reveals r?

Exercise. What happens if Alice uses the same r in two different sessions?

Soundness proof. Let $\mathcal{A}$ be an adversary running in time t that on input (p, q, g, h) can impersonate Alice with probability at least ε.

$\mathcal{A}$ runs in two phases, $\mathcal{A}_1$ and $\mathcal{A}_2$.

- $\mathcal{A}_1$ outputs the commitment x.
- $\mathcal{A}_2$, on input of the challenge k, outputs the response y.
- $\mathcal{A}_1$ also outputs some "internal state" s that is input to $\mathcal{A}_2$.

Construct an algorithm $\mathcal{B}$ as follows:

1. Let $(x, s) \leftarrow \mathcal{A}_1(p, q, g, h)$.
2. Let $\mathbf{K} = \{1, \ldots, 2^\eta\}$. Repeat:
   (a) Randomly choose a challenge $k_1 \in \mathbf{K}$.
   (b) Get the response $y_1 \leftarrow \mathcal{A}_2(k_1, s)$.

   Until verification succeeds.
3. Let $\mathbf{K} = \{1, \ldots, 2^\eta\} \backslash \{k_1\}$. Generate k_2, y_2 as in previous step.
4. Use these values to find $a = \log_g h$. **Exercise:** how?

Average running time of $\mathcal{B}$:

- Step 1: $\leqslant t$.
- Step 2: $\leqslant t/\varepsilon$.
- Step 3: $\leqslant \frac{t}{\varepsilon - 2^{-\eta}}$.
- Step 4: Some.

Hence the existence of the algorithm $\mathcal{A}$ allows us to take discrete logarithms in time $O(\frac{t}{\varepsilon - 2^{-\eta}})$.

This is efficient if ε is larger than $(1 + o(1)) \cdot 2^{-\eta}$. $\square$

(Recall that it is trivial to construct an adversary whose success probability is $2^{-\eta}$).

What does Bob learn after running Schnorr's identification protocol?

- That the other party knew Alice's secret key.
- Anything else???

Could Bob impersonate Alice afterwards? We did not prove that the protocol does not leak Alice's secret key a...

No proof of such kind of property is known for Schnorr's scheme.

We only proved that before interacting with Alice, Bob cannot impersonate her.

Okamoto identification scheme. Let η, p and q be as before. Let $g_1, g_2 \in \mathbb{Z}_p^*$ be two elements of order q, such that nobody knows $c = \log_{g_1} g_2$.

Alice's secret key: $(a_1, a_2) \in \mathbb{Z}_q^2$. Public key: $h = g_1^{a_1} g_2^{a_2}$.

Commitment Alice picks random $r_1, r_2 \in \mathbb{Z}_q$, sends $x = g_1^{r_1} g_2^{r_2}$ to Bob.

Challenge Bob picks a random $k \in \{1, \ldots, 2^\eta\}$ and sends it to Alice.

Response Alice sends $y_1 = (r_1 + a_1 k) \bmod q$ and $y_2 = (r_2 + a_2 k) \bmod q$ to Bob.

Verification Bob verifies that $x = g_1^{y_1} g_2^{y_2} / h^k$.

Exercise. Show that the scheme works.

SECURITY PROOF. Suppose that the adversary $\mathcal{A}$ is able to impersonate as Alice (with probability $\varepsilon \geqslant (1+o(1)) \cdot 2^{-\eta}$). The adversary may have run the protocol with Alice.

Similarly to the proof for Schnorr's scheme, we can find some $(b_1, b_2) \in \mathbb{Z}_q$, such that $h = g_1^{b_1} g_2^{b_2}$.

(b_1, b_2) is most probably different from (a_1, a_2).

- Because $\mathcal{A}$-s view on the protocol depends only on $a_1 + ca_2 = b_1 + cb_2$, not on the whole (a_1, a_2).

From $g_1^{a_1} g_2^{a_2} = g_1^{b_1} g_2^{b_2}$ we find $c = \log_{g_1} g_2$. $\square$

Exercise. Consider the following identification scheme:

Alice's secret key: large primes p and q (such that $p \equiv q \equiv 3 \pmod 4$).

Public key: $n = pq$.

Challenge Bob picks some $z \in \mathbb{Z}_n$ and sends $y = z^2 \bmod n$ to Alice.

Response Alice finds $x \in \mathbb{Z}_n$, such that $x^2 \equiv y \pmod n$, and sends x to Bob.

Verification Bob checks that $x^2 = y$.

Why is this scheme insecure?

Fiat-Shamir identification scheme.

- Key generation: Alice generates two large primes p, q and computes $n = pq$. Alice generates a random $s \in \mathbb{Z}_n^*$ and computes $v = s^2 \bmod n$.
  - Public key: (n, v). Secret key: (n, s).
- Protocol:

  Commitment Alice generates a random $r \in \mathbb{Z}_n \setminus \{0\}$, computes $x = r^2 \bmod n$, and sends x to Bob.

  Challenge Bob generates a random $b \in \{0, 1\}$ and sends it to Alice.

  Response Alice sends $y = rs^b \bmod n$ to Bob.

  Verification Bob accepts if $y^2 = xv^b$.

If Alice knows s then she can always make Bob accept by computing y correctly.

If the adversary can compute s from (n, v) then he can also factor n. This is supposedly intractable.

SOUNDNESS: How successfully can the adversary impersonate Alice without knowing s?

The adversary cannot respond correctly to both challenges (0 and 1).

If he knows both r and rs then he can compute s.

If the adversary can correctly guess b that Bob is going to send then he may

- Choose $y \in \mathbb{Z}_n \setminus \{0\}$ and compute $x = y^2 \cdot v^{-b} \bmod n$. Use that x as the commitment.

- y will then be the correct response.

Hence the adversary can fool Bob only with probability 50%.

Executing the protocol several times will exponentially diminish that probability.

Exercise. If the challenge b was not chosen from the set $\{0, 1\}$, but from the set $\{0, \ldots, m - 1\}$, then what would the fooling probability be?

SECURITY:

What does Bob (or an adversary) "learn" from an execution of that protocol?

Well, whatever...

But the "new information" is certainly upper-bounded by

- Bob's random choices;
- the trace (x, b, y) of the protocol.

Here (x, b, y) is generated according to a distribution where

- x is a random quadratic residue modulo n;
- b is a random bit;
  - Its distribution may depend on x.
  - I.e. Bob may be actively trying to determine Alice's secret s.
- y is a square root of xv^b.
  - $y = rs^b$ is a random element of $\mathbb{Z}_n \backslash \{0\}$ because r is a random element of $\mathbb{Z}_n \backslash \{0\}$ and s is invertible in $\mathbb{Z}_n$.

Bob (or anyone else) can sample this distribution himself:

- Generate a random bit b^* by tossing a fair coin.
- Generate a random $y \in \mathbb{Z}_n \backslash \{0\}$.
- Set $x = y^2 v^{-b^*} \bmod n$.
- Generate the random bit b according to the distribution that depends on x.
- If $b \neq b^*$ then start over.

We see that all "new information" that Bob could obtain by running the protocol could have been generated by Bob himself, without the help of Alice.

Hence there really was no new information (beside the fact that Alice knows the secret key).

We say that this protocol has the property of zero-knowledge (*nullteadmus*).

This was an example of a zero-knowledge proof of knowledge.

Let G be a cyclic group where taking discrete logarithms is hard, let g be a generator of G and $m = |G|$. Let Alice generate $a \in \mathbb{Z}_m$ and publish $h = g^a$.

Alice can prove her knowledge of a to Bob as follows:

Commitment Alice generates a random $r \in \mathbb{Z}_m$, computes $x = g^r$ and sends x to Bob.

Challenge Bob generates a random $b \in \{0, 1\}$ and sends it to Alice.

Response Alice sends $y = r + ab$ to Bob.

Verification Bob accepts if $g^y = xh^b$.

Exercise. Prove that the protocol works, is secure, and has the zero-knowledge property.

Several rounds of the protocol have to be run, such that the probability of Alice not cheating is high enough.

They may be run one after another or in parallel.

Or can they?

Exercise. What is the difference between running rounds one after another and running them in parallel?

Recall the simulation (for a single round):

- Generate $b^* \in \{0, 1\}$ by tossing a fair coin.
- $\ldots$
- Obtain $b \in \{0, 1\}$; its distribution depends on things that happened above.
- If $b \neq b^*$ then start over.

Probability of succeeding (not starting over): $1/2$.

To simulate k rounds, we have to do the work above approximately $2k$ times.

For k rounds the simulation would be

- Generate $b_1^*, \ldots, b_k^* \in \{0,1\}$ by tossing fair coins.
- $\ldots$
- Obtain $b_1, \ldots, b_k \in \{0,1\}$; their distribution depends on things that happened above.
- If $\exists i : b_i \neq b_i^*$ then start over.

Probability of succeeding: $1/2^k$.

Exponentially small in k.

To simulate k rounds, we have to do the work above approximately 2^k times.

We have seen zero-knowledge proofs of knowledge of factorization and discrete log.

Proof of knowledge $\Leftrightarrow$ identification scheme.

We also saw a non-zero-knowledge proof of knowledge (of a secret key of some asymmetric cryptosystem).

Consider now the case where the Prover

- knows that a certain claim holds;
- knows its proof;
- wants to convince Verifier that the claim holds;
- does not want to reveal anything else.

For example, Prover wants to convince Verifier that (g, h, y_1, y_2) is a Diffie-Hellman tuple (here $g, h, y_1, y_2 \in G$ for some group G, $m = |G|$).

I.e. $\exists x \in \mathbb{Z}_m$ (which Prover knows) such that $y_1 = g^x$ and $y_2 = h^x$.

Recall our "voting scheme":

- There are a number of voters $V_1, \ldots, V_k$.
- The voter V_i has a choice $e_i \in \{0, 1\}$.
- The Tallier has an ElGamal public key h. He knows a, such that $g^a = h$.
- The voter V_i generates a random r_i and publishes $(g^{r_i}, g^{e_i} h^{r_i})$.
- The votes are multiplied, resulting in $(g^R, g^E h^R) = (c_1, c_2)$, where $E = \sum_i e_i$.
- The Tallier decrypts, and publishes g^E. Brute-forcing reveals E.

Tallier acted correctly if $(g, c_1, h, c_2 g^{-E})$ is a Diffie-Hellman tuple. The common exponent is a.

Prover and Verifier know G, m, (g, h, y_1, y_2).

Prover knows x, such that $g^x = y_1$, $h^x = y_2$.

Commitment Prover randomly picks $r \in \mathbb{Z}_m$ and sends $A = g^r$ and $B = h^r$ to Verifier.

Challenge Verifier sends a random bit $b \in \{0, 1\}$ to Prover.

Response Prover sends $s = (r + bx) \bmod m$ to Verifier.

Verification Verifier accepts if $A = g^s y_1^{-b}$ and $B = h^s y_2^{-b}$.

Exercise. Prove that the protocol works, is secure, and has the zero-knowledge property.

The protocol may be understood as follows:

The Prover made the following claims:

0. A and B are constructed correctly (i.e. $\log_g A = \log_h B$).
1. If A and B are constructed correctly then $\log_g y_1 = \log_h y_2$.
   - $y_1 = g^{s - \log_g A}$ and $y_2 = h^{s - \log_h B} = h^{s - \log_g A}$.

The Verifier will verify one of these claims, but the Prover does not know beforehand, which one.

This was an example of zero-knowledge proof.

Let us have protocols for proving in zero knowledge that A_0 holds, and that A_1 holds.

Then there is a protocol for proving in zero knowledge that $A_0 \vee A_1$ holds.

With this protocol, a voter can prove the correctness of his vote.

Suppose that A_i holds and the prover is able to prove it in zero knowledge.

Commitment Run the simulator for A_{1-i}, producing $(c_{1-i}, b_{1-i}, r_{1-i})$. Generate c_i as when proving A_i. Send (c_0, c_1) to the verifier.

Challenge The verifier chooses a bit b and sends it to the prover.

Response Set $b_i = b \oplus b_{1-i}$. Generate r_i as when proving A_i, using b_i as the challenge. Send (b_0, b_1, r_0, r_1) to the verifier.

Verification Check that (c_0, b_0, r_0) is a proof for A_0, (c_1, b_1, r_1) is a proof for A_1 and $b_0 \oplus b_1 = b$.

Let us play the following game. We both choose a bit. If their xor is 1 then you win, otherwise I win.

- So, what is your bit?
- . . .
- Tough luck, so is mine.

This seems to be unfair. . .

- So, what is your bit?
- My bit? It is in that sealed envelope. What is yours?
- My bit is. . .
- OK, you may open the envelope now.

This is fair.

The envelope was an example of bit commitment (*bitikinnistus*).

A bit commitment is a cryptographic primitive with three operations:

- Key generation;
- Committing — takes the secret key and the bit to be commited, and produces the commitment and the revealing information.
- Verifying — takes the public key, commitment, the bit that was allegedly commited, and revealing information, and either accepts or rejects.

A bit commitment must have two properties:

Concealing The public key and commitment should not reveal the committed bit.

Binding It must be impossible to produce a commitment that can be opened both ways.

Historically, encryption has been used for commitment.

- To commit, generate a new key K and a random string R.
- Commitment of b is $E_K(f(b, R))$ for some f that combines b and R.
- Revealing information is (K, R).
- Verification: recompute $E_K(f(b, R))$.

Concealing is obvious. Binding depends on E and f.

Bit-commitment based on quadratic residuosity:

Key generation Let $p, q \in \mathbb{P}$, $n = pq$, $m \in \mathbb{Z}_n$, such that $\left(\frac{m}{p}\right) = \left(\frac{m}{q}\right) = -1$. (n, m) is the public key.

- Then $\left(\frac{m}{n}\right) = 1$, but m is a quadratic non-residue modulo n.

Committing Choose a random $x \in \mathbb{Z}_n$. The commitment is $c = m^b x^2 \bmod n$. The revealing information is x.

Verifying Check whether $c \equiv m^b x^2 \pmod{n}$.

The scheme is unconditionally binding because the commitments of 0 are quadratic residues, and the commitments of 1 quadratic non-residues.

It is believed that distinguishing quadratic residues from non-residues is hard. Under this assumption, the scheme is concealing.

Exercise. n and m are generated by the Prover. What happens if the Prover lets m to be a quadratic residue?

Another one:

Key generation Let $p, q \in \mathbb{P}$, $n = pq$. Committer must not know p and q (recipient may know them). Let m be a quadratic residue modulo n. (n, m) is the public key.

Committing Choose a random $x \in \mathbb{Z}_n$. The commitment is $c = m^b x^2 \bmod n$. The revealing information is x.

Verifying Check whether $c \equiv m^b x^2 \pmod{n}$.

Concealing is unconditional — the possible commitments are the same for 0 and 1.

If a committer could open c as both 0 and 1, then he knows x_0 and x_1, such that

$$x_0^2 = c = m x_1^2 \quad .$$

Then $m = \frac{x_1^2}{x_0^2}$ and $\sqrt{m} = x_1/x_0$. I.e. the committer can compute square roots modulo n. Hence he can also factor n.

We have seen two schemes.

One was computationally concealing, but unconditionally binding.

The other was uncondtionally concealing, but only computationally binding.

Exercise. Are there schemes where both concealing and hiding are unconditional?

Commitments can be used to give zero-knowledge proofs for any problems in **NP**.

Example: graph 3-colourability (NP-complete).

Given a graph (V, E). The Prover knows how to colour its vertices with three colours, such that no edge has both endpoints of the same colour.

Let $\varphi : V \to \{1, 2, 3\}$ be the colouring.

The Prover wishes to communicate the 3-colourability of (V, E) to the Verifier, without giving away φ.

Let $V = \{v_1, \ldots, v_n\}$ and $E \subseteq V \times V$. The prover

- Chooses a random permutation π of the set $\{1, 2, 3\}$;
- Lets c_i be a commitment to $\pi(\varphi(v_i))$ $(1 \leqslant i \leqslant n)$;
  - To commit to a several bits long value, commit to each bit separately.
- Sends $(v_1, c_1), \ldots, (v_n, c_n)$ to the Verifier.

(The Commitment)

The Verifier picks an edge (v_i, v_j) and sends it to the Prover. (The Challenge)

The Prover opens the commitments c_i and c_j. (The Response)

The Verifier checks that the colours for v_i and v_j are different.

If the graph (V, E) is not 3-colourable then there exists at least one edge having the endpoints of the same colour.

An honest Verifier finds it with the probability $\geqslant 1/m$.

The probability that a Verifier is fooled after k rounds is at most $\left(1 - \frac{1}{m}\right)^k$.

If we take $k = m^2$ (polynomial in the size of the graph) then this probability is about e^{-m}.

Because $\lim_{m\to\infty} \left(1 - \frac{1}{m}\right)^m = 1/e$.

Hence the protocol is secure.

It is obvious that the protocol works.

How to construct transcripts without the Prover?

First, select the challenge (v_i, v_j).

Let c_i and c_j be commitments to different colours. Let the committed colours of other vertices be random.

Note that the resulting distribution is not the same as the real one (using the Prover), but it is indistinguishable from that.

This is an example of computational zero-knowledge. If the distributions are equal then we have perfect zero-knowledge.

Example: Graph isomorphism in perfect zero knowledge.

Given two graphs $G_0 = (V_0, E_0)$ and $G_1 = (V_1, E_1)$. The Prover knows a graph isomorphism $\varphi : V_0 \longrightarrow V_1$.

The Prover wants to convince the Verifier that $G_0 \cong G_1$.

Commitment. Prover generates $G_1' = (V_1, E_1')$ as a random isomorphic copy of G_1 and sends it to the Verifier.

I.e. The Prover selects a random permutation ψ of V_1 and takes

$$E_1' = \{(\psi(u), \psi(v)) \,|\, (u,v) \in E_1\} \ .$$

Challenge. The Verifier sends a random bit $b \in \{0,1\}$ to the Prover.

Response. If $b = 1$ then Prover returns $f = \psi$. If $b = 0$ then Prover returns $f = \psi \circ \varphi$.

Verification. The Verifier checks that f is an isomorphism from G_b to G_1'.

Simulation (for an honest Verifier).

First generate $b \in \{0, 1\}$.

Then generate G_1' as a random isomorphic copy of G_b.

For a dishonest verifier, generate $b^* \in \{0, 1\}$ whose distribution may depend on G_b. If $b \neq b^*$ then start over.

Parallel composition of k sessions for graph 3-colourability:

1. Prover sends the commitments $C_1, \dots, C_k$;
2. Verifier replies with the challenges $b_1, \dots, b_k$;
3. Prover sends the responses $r_1, \dots, r_k$;
4. Verifier checks that C_i, b_i, r_i are correctly related.

Problem: b_i may depend on C_j for $j > i$ and the simulation is no longer expected polynomial-time.

How about:

1. Verifier sends the challenges $b_1, \ldots, b_k$;
2. Prover sends the commitments $C_1, \ldots, C_k$;
3. Prover sends the responses $r_1, \ldots, r_k$;
4. Verifier checks that C_i, b_i, r_i are correctly related.

Well. . .

That does not prove anything anymore. . .

How about this:

1. Verifier sends the bit-commitments $c_1, \ldots, c_k$ to the challenges, using an unconditionally concealing commitment scheme;

2. Prover sends the commitments $C_1, \ldots, C_k$ using an unconditionally binding commitment scheme;

3. Verifier opens $c_1, \ldots, c_k$; Prover learns $b_1, \ldots, b_k$;

4. Prover sends the responses $r_1, \ldots, r_k$;

5. Verifier checks that C_i, b_i, r_i are correctly related.

Here C_i may depend on $c_i \ldots$ but this dependence does not help the prover in choosing the opening r_i of C_i.

A (zero-knowledge) proof is a protocol.

It is interactive.

Can we make it non-interactive?

I.e. the prover sends a single message to the verifier and the verifier is convinced (or not).

A common way is:

- Let $h : \{0,1\}^* \to \{0,1\}^k$ be a “secure” hash function.
- The prover generates commitments $C_1, \ldots, C_k$;
- let $b_1 \cdots b_k = h(C_1, C_2, \ldots, C_k)$;
- The prover generates responses r_i for C_i and challenge b_i.

The whole proof is $((C_1, r_1), \ldots, (C_k, r_k))$.

The verifier regenerates $b_1, \ldots, b_k$ and verifies all k rounds.

k must be long enough, such that regenerating $C_1, \ldots, C_k$ until we get right challenges is infeasible.

h must be a (pseudo)random function. But h is public. h is a random oracle.

Also, an identification scheme may be converted to a signature scheme.

To sign message m:

- Generate a commitment C;
- Let $k = h(m, C)$
- Construct a response r for the commitment C and challenge k.
- Signature is (C, r).

k must be long enough to prevent regenerating C until a right challenge is found.

Schnorr's signature scheme is defined this way.