Commitments – a commitment scheme allows a prover to choose a value without revealing it.
The prover can later reveal her choice ('open' the commitment) and the verifier can use the commitment to check that the revealed choice is correct.
A building block for ZK protocols or as a primitive. A commitment scheme can be used for coin flipping over the telephone.

A commitment scheme has two phases:
Commitment phase: The prover P commits to the value x and gives the commitment $C(x)$ to the verifier V.
Opening phase: The prover P send additional information to V which allows V to compute x.

The commitment may be randomized in which case a random value r is included in the input and we write $C(x, r)$.

Hiding and binding – a commitment scheme should have two properties:
Hiding: The verifier should not be able to find the value x during the commitment phase.
Binding: The prover should not be able to change the value x after the commitment phase.

Hiding and binding may be unconditional (unbreakable independent of available computing power) or computational (breakable by enough computing power).
Hiding and binding cannot both be unconditional.

Examples

1. $C(x) = g^x$ where $x \in \mathbb{Z}_q$ and g generates a subgroup of $\mathbb{Z}_p^*$ of order q and q is a large prime. In the opening phase P gives x to V.
2. $C(x, r) = g^x h^r$ where $x \in \mathbb{Z}_q$. Both g and h generate a subgroup of $\mathbb{Z}_p^*$ of order q and q is a large prime. The random value r is chosen in $\mathbb{Z}_q$.
   - The value of $\log_h g$ must be unknown. In the opening phas P gives x and r to V.
   - This is known as *Pedersen's scheme*.

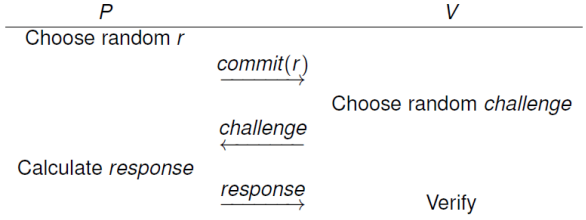
1) Conditional hiding and Unconditional binding

2) Unconditionally hiding and conditionally binding

Proof of knowledge:
Zero knowledge (ZK) protocols are interactive proof systems between two entities called the prover P and the verifier V.
We only consider proof of knowledge where P aims to convince V that she holds a secret value satisfying some property.
One main application of proofs of knowledge is in identification protocols in which the prover demonstrates ownership of an identifying secret.
ZK protocols have been used in a variety of applications including anonymous cash, voting and escrow.

Typical interactive protocol:

- An interactive protocol consists of a sequence of messages sent between P and V



- The sequence of messages sent in a particular protocol instance is called a *transcript*.

Protocol properties – there are three fundamental properties that interactive protocols may satisfy:
1. Completeness – If P and V act correctly then V accepts the claim (except with negligible property).
2. Soundness – A dishonest prover P' (who doesn't know the secret value) cannot successfully execute the protocol (except with negligible probability).
3. Zero knowledge – The protocol is simulatable. There is an efficient algorithm to produce protocol transcripts, without interaction with the real prover, that are indistinguishable from those made with the real prover.

Types of zero knowledge:
Honest verifier ZK assumes that V acts correctly – chooses inputs randomly when required. This is often much easier to prove than full ZK, and in many cases is good enough.

Perfect ZK requires that transcript from the simulator are indistinguishable no matter how powerful the distinguishing algorithm is.

Computational ZK only requires that protocol transcript from the simulator are indistinguishable by an efficient polynomial time algorithm.

Protocol simulation:
The simulator need not generate the transcripts in the same order as the protocol. In fact this cannot be the case, since the simulator does not have the secret value.
In order to be perfect zero knowledge, it must be possible to simulate transcripts whatever the strategy used by the verifier. The verifier may choose a challenge that depends on the commitment.
Therefore, a simulator should be defined to work with an arbitrary verifier V^* which is any (probabilistic) efficient algorithm.

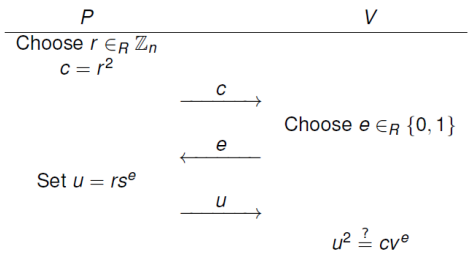
Types of soundness:
Statistical soundness requires that a prover can't cheat no matter how computational powerful it is.
Computational soundness only requires the protocol that cheating is not possible by an efficient prover algorithm.

Generally, it's not possible to achieve perfect ZK and perfect soundness.
A ZK scheme with computation soundness is often called an argument and not a proof.

Fiat-Shamir identification protocol:
Uses a modulus $n = pq$ for two large prime numbers p and q as in the RSA cryptosystem.
An important difference from RSA is that all users can have the same value of n since nobody needs to know the value of p and q. Once a suitable n is chosen the factors can be destroyed permanently.
The security of the protocol is based on the difficulty of taking square roots of modulo n.
Rabin (some guy) proved this problem to be as hard as factorizing n.

Fiat-Shamir – key generation:
P can choose her own secret key S randomly in the range $0 < s < n$.
P then calculates her public key $v = s^2 \ mod \ n$. Thus finding the secret key from the public key alone is as hard as factorizing n.
P may then arrange to have her public key certified by a suitable authority.
The purpose of the protocol is for P to convince V that she knows the secret key s and is the true owner of the certificate.

Fiat-Shamir - identification protocol (one round)



The protocol must be repeated multiple times

Fiat-Shamir - soundness

- Consider a cheating prover P' who can reliably answer challenge c for both $e = 1$ and $e = 0$
- So P' can compute u and $\tilde{u}$ with

$$u^2 = c$$
$$\tilde{u}^2 = cv$$

Then $(u/\tilde{u})^2 = v$ and so $(u/\tilde{u})$ is (equivalent to) the secret s.

- Thus P' can extract the secret if it can run the protocol successfully

Fiat-Shamir – soundness:
If P' can guess the value of e it can always find a valid u.
The identification protocol must be run multiple times in order to reduce the probability of cheaters.

Fiat-Shamir is honest verifier ZK:
To show that the Fiat-Shamir protocol is honest verifier ZK, we need a simulator when V chooses e as a random bit. Transcripts of the protocol consists of triples (c, e, u), where c is a random square modulo n, e is arandom bit and $u^2 = cv^e$

A simulator generates identically distributed triplets:
1. Choose a random integer r and a random bit e.
2. Set $c = \frac{r^2}{v^e}$
3. Set $u = r$
4. Output (c, e, u)

Fiat-Shamir is perfect ZK:
To construct a general simulator, let V^*be any efficient verifier algorithm. V^* returns a bit e^*given any challenge c.
1. Choose random integer r and random bit e.
2. Set $c = \frac{r^2}{v^e}$
3. Give c to V^* and obtain output e^*
4. If $e = e^*$ then set $u = r$ and output the triplet (c, e, u). Otherwise return to step 1.

It's impossible to distinguish simulated transcript from transcripts generated when the genuine P is following the protocol with V.

Efficiency of a simulator:
The probability that e and e* are the same bit is 50%, independently of how V^* chooses e^*.
Therefore the algorithm succeeds each time with probability 50% and so the expected time to get one triplet is two iterations. The expected time to get k triplets is $2k$ iterations.

Indistinguishability – To see that the transcripts are indistinguishable from the real transcripts.
1. Note that the c values are uniformly distributed over all squares of modulo n.
2. Notice that whether a particular c is output is independent of the value of e (since c can equally be chosen as r^2 or $\frac{r^2}{v}$)
3. Therefore the c values accepted are random squares.

Computational requirements:
Fiat-Shamir identification is an alternative to using simple challenge-response protocols.
An example is where a random challenge is signed in a response. Such a protocol is not ZK and much more computationally expensive.
RSA signature generation requires on average 3072 modular multiplications using a 2048-bit modulus.
The Fiat-Shamir protocol is much more efficient, requiring only 30 modular multiplications.

Extensions and variants: A parallel version of the protocol allows the amount of interaction to be reduced at the cost of a reduced security guarantee.
A digital signature can be based on the Fiat-Shamir protocol by using a hash function to "choose" the randomness.

Proofs of knowledge for discrete logarithms – The basic example is a proof that the verifier knows the discrete logarithm of a specified value, such as its public key.
A generalized version of the same protocol is used to show the equivalence of two or more discrete logarithms to different bases. In the following we assume that p is a large prime, q is a prime with $q|p-1$, and g is an element in Z_p^* of order q. All computations take place in Z_p^*.

Protocol

- P shows knowledge of the discrete log, x, of h to the base g – that $h = g^x$.

P		V
Choose $s \in_R \mathbb{Z}_q$ Set $a = g^s$	$\xrightarrow{a}$	
		Choose $c \in_R \mathbb{Z}_q$
	$\xleftarrow{c}$	
Set $r = s + cx \bmod q$	$\xrightarrow{r}$	
		$g^r h^c = a$

Completeness: this follows by checking that the equations are satisfied if the protocol is run correctly.
Soundness: Having made a commitment to a, if P is able to respond correctly to two or more challenges then she must know x.
Zero knowledge: This protocol is honest verifier ZK but not know to be proper ZK.

Non-interactive zero knowledge (NIZK) – interaction seems to be critical for ZK since P can't be allowed to know the challenge in advance. There are two known methods for achieving NIZK where P doesn't know the challenge c until the commitment t is made.
1. Use a hash function, modelled as a random function, to compute the challenge based on a commitment.
2. Use public randomness to create a common reference string (CRS) which can't be influence by the prover.

The first method is often called the
Fiat-Shamir heuristic: THIS IS WHAT THE "PROTOCOL image, previous page" IS ALL ABOUT.
Peggy (P) has a password x and hashes it with generator g:

$$y = g^x,$$

and sends it to Victor (V).
Peggy selects a random v, such that transcript

$$t = g^v,$$

and sends it to Victor.
Victor generates a random value c based on t and message m, and sends it to Peggy.
Peggy calculates

$$r = v - cx,$$

and sends it to Victor.
Victor computes

$$g^r y^c = g^{v-cx}\,(g^x)^c = g^{v-cx}g^{xc} = g^{v-cx+cx} = g^v = t$$

and Peggy is verified.

This works because Peggy knows her password x and the random value v, and she's the only one who'd generate the correct value of r.

The triple (t, c, r) represents the transcript and can be used as a signature.

Signatures from proof of knowledge – the proof of knowledge of discrete logs can be made into a normal digital signature with the Fiat-Shamir heuristic. See example above.

Schnorr signature – constructed by applying the Fiat-Shamir heuristic to Schnorr's identification protocol.

Generation: compute the triple (t, c, r) as above. The signature of m is the pair (c, r).
Verification: Given the claimed signature (c, r) and message m, calculate $t' = g^r y^{-c}$.
Check that $c = H(t', m)$, and if so accept the signature.

Zero knowledge in cryptocurrencies:
Zerocoin uses a proof of knowledge made non-interactive with Fiat-Shamir heuristic.
Zcash uses zk-**SNARK**s: zero knowledge **S**uccinct **N**on-interactive **AR**guments of **K**nowledge.
zk-SNARKs allow proof of multiple constraints with constant sized proofs and efficient verification.

Zerocash is abstract academic proposal with formal models and security proof.
User can mint zcoins of any value from underlying currency.
"Minting a Zerocoin means destroying a fixed number of Zcoin and turning them into 'Zerocoins' which are tokens to redeem completely new 'clean' Zcoins."
User can pour minted zcoins anonymously into new coins of same value to any Zerocash address. Allows both splitting and merging of minted zcoins. User can receive coins into value of the underlying currency.

Use of zk-SNARKs:
zk-SNARKs are compact proofs of any constraints expressed as an arithmetic circuit.
Constraints required in Zerocash include:
- inputs are valid (appear on the ledger).
- the correct signing key is used.
- a new serial number was correctly computed from the old serial number.
- new and old coins are properly constructed.
- balance is preserved.

The zerocash protocol includes a zk-SNARK in each pour operation to guarantee that anonymous transactions are valid.

Zcash parameter generation:
The common reference string used for constructing the zk-SNARK parameters requires a trusted process. Parameters were generated in a complex multi-party ceremony to ensure that no party knows the secret inputs. Much more complex than Zerocoin parameters.

Comparing anonymity properties:
Coinjoin: anonymity set is the set of inputs to the shared transaction.
Zerocoin: anonymity set is the set of users how have minted zerocoins.
Zerocash: anonymity set is any user with a valid address.

Merged mining:
Mining happens in parallel between main currency (typically Bitcoin) and an altcoin.
The altcoin is designed to use Bitcoin coinbase transaction to store its Merkle root of altcoin transactions. To validate an altcoin block, need to check inputs to Bitcoin header such that hash is below the altcoin target (usually less or equal to Bitcoin target). Miners get to mine altcoins 'for free'.
No change to Bitcoin is needed – miners simply choose to merge mine by including the altcoin details.
Multiple altcoins can be merge mined simultaneously.

Merge mining pros and cons:
Merge mining can help altcoins bootstrap with high difficulty.
This can increase security at the start of the altcoin's life.
Marginal cost to miner is mainly transaction validation. It can lead to concentration of power since small miners have same cost. May lead to unvalidated transactions being mined.

Atomic cross-chain swaps:
Exchanging value in different chains without a trusted third party.
Suppose A wants to swap altcoin value with B who has Bitcoin value.
Use multisigs and time locks as in fair exchange problems.
Broadcast means that a transaction is posted to its relevant blockchain.
A choses random x and computes $H(x) = R$

Atomic cross chain swaps – A setup altcoin
A makes deposit transaction, D_A to script S_A which can be redeemed with either $multisig(A, B)$ or $sig(B)$ and value x with value $H(x) = R$
A makes refund transaction, R_A, paying value from S_A back to A, timelocked to T_2
A gets $sig(B)$ on R_A and then broadcasts D_A

Atomic cross-chain swaps – B setup on Bitcoin
B makes deposit transaction, D_B, to script S_B which can be redeemed with either $multisig(A, B)$ or $sig(A)$ and value x with value $H(x) = R$
B makes refund transaction, R_B, paying value from S_B back to A, timelocked to T_1 with $T_1 < T_2$
B gets $sig(A)$ on R_B and then broadcasts D_B

Atomic cross-chain swaps – settlement
A can claim Bitcoin value before T_1 by revealing x
Then B can claim altcoin value before T_2
If A does not reveal x before T_1 then B can claim refund at time T_1
Then A can claim refund at time T_2