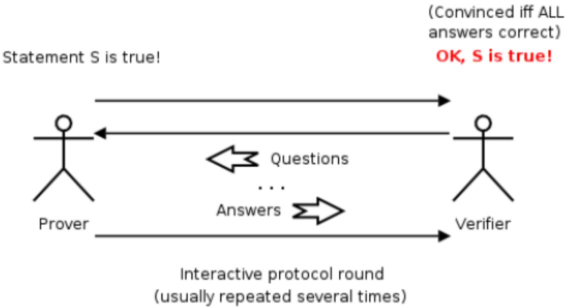
IT486: Blockchains and Cryptocurrencies

Zero-knowledge range proofs

An interactive proof system

Examples of statements:

- I am Peggy (for identification)
- I have the secret key for this public key



Properties that zero-knowledge proofs must have

- Completeness
  - if statement is true, honest verifier will eventually be convinced by honest prover

Properties that zero-knowledge proofs must have

- Completeness
  - if statement is true, honest verifier will eventually be convinced by honest prover
- Soundness
  - if statement is false, no (cheating) prover can convince verifier that it is true (except with small probability)

Properties that zero-knowledge proofs must have

- Completeness
  - if statement is true, honest verifier will eventually be convinced by honest prover
- Soundness
  - if statement is false, no (cheating) prover can convince verifier that it is true (except with small probability)
- Zero-knowledge
  - if statement is true, no (cheating) verifier learns anything other than this fact. Verifier cannot even prove this fact to anyone later

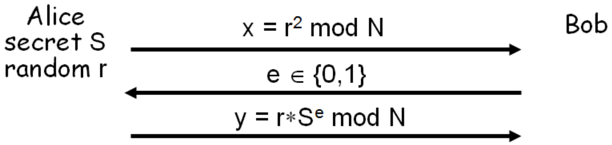
Range proofs

- Prover tries to convince a verifier that a certain encrypted value x lies in a given range $[a, b]$ without revealing any information on x besides that it lies in the given range
- Example: Proving that the transaction amount is non-negative

Fiat-Shamir ZKP

- Suppose $N = pq$, where p and q prime
- Computational assumption:
  - Finding square roots modulo N when p, q are kept secret is hard
- Let $v = S^2 \bmod N$. Revealing v does not reveal S
- Goal: Assume Alice knows S. She must convince Bob that she knows S without revealing any information about S

Fiat-Shamir ZKP



- Public: Modulus N and $v = S^2 \bmod N$
- Alice selects random r
- Bob chooses $e \in \{0, 1\}$
- Bob verifies that $y^2 = r^2 \cdot S^{2e} = r^2 \cdot (S^2)^e = x \cdot v^e \bmod N$

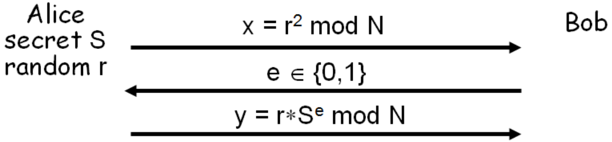
Can Bob find S?

- Public: $v = S^2 \bmod N$
- If Bob can find modular square roots, he can get S from public v

Can Bob find S?

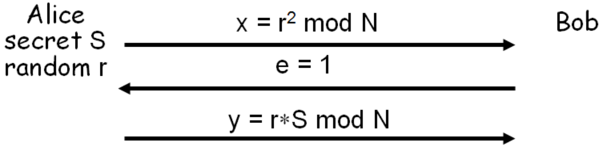
- Bob sees $r^2 \bmod N$ in message 1
- Bob sees $r \cdot S \bmod N$ in message 3 (if $e = 1$)
- If Bob can find r from $r^2 \bmod N$, he gets S. But that requires modular square root

Can Bob find S?



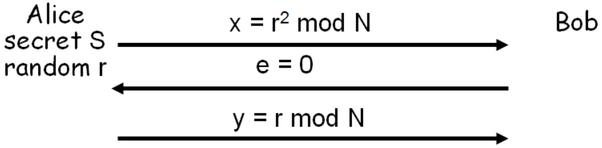
- Alice must use new r each iteration or else
- If $e = 0$, Alice sends r in message 3
- If $e = 1$, Alice sends $r \cdot S$ in message 3
- Anyone can find S given both r and $r \cdot S$

Protocol run with $e = 1$



- Public: Modulus N and $v = S^2 \bmod N$
- Alice selects random r
- Suppose Bob chooses $e = 1$
- Bob must verify that $y^2 = x \cdot v \bmod N$
- Alice must know S in this case

Protocol run with $e = 0$



- Public: Modulus N and $v = S^2 \bmod N$
- Alice selects random r
- Suppose Bob chooses $e = 0$
- Bob must verify that $y^2 = x \bmod N$
- Alice does not need to know S in this case!

Soundness property fulfilled?

- Suppose Alice does not know the secret S
- If Alice expects Bob to send $e = 0$, she can send $x = r^2$ in msg 1 and $y = r$ in msg 3 (i.e., follow protocol)
- If Alice expects Bob to send $e = 1$, she can send $x = r^2 v^{-1}$ in msg 1 and $y = r$ in msg 3 (i.e., disobey protocol)

Soundness property fulfilled?

- Alice can fool Bob with prob $1/2$, but ...
- after n iterations, the probability that Alice can fool Bob is only $1/2^n$
- Bob's $e \in \{0, 1\}$ must be unpredictable

Zero-knowledge property fulfilled?

- Can Bob forge a transcript on his own, without interacting with Alice?
- Bob's transcript will consist of a sequence of three-message rounds of the form:

$$\begin{aligned}
A \rightarrow B &: \quad x_1 \\
B \rightarrow A &: \quad e_1 \\
A \rightarrow B &: \quad y_1 \\
A \rightarrow B &: \quad x_2 \\
B \rightarrow A &: \quad e_2 \\
A \rightarrow B &: \quad y_2 \\
\ldots
\end{aligned}$$

Zero-knowledge property fulfilled?

- Whatever Bob might be able to do after actually taking part in the protocol he could equally well do by just using a forged transcript
- Hence Bob doesn't gain any additional knowledge!

Proof of Knowledge of Exponent

- Group G of prime order q with generator g
- Let $x \in Z_q$ be a secret quantity and let $X = g^x$
- Prover is able to prove her knowledge of x to Verifier
- No information about x is gained by Verifier

Schnorr's Protocol

1. Prover chooses $r \in Z_q$ randomly and hands $R = g^r$ to Verifier
2. Verifier chooses $c \in Z_q$ randomly and hands it to the Prover
3. Prover computes $s = cx + r \pmod q$ and hands s to the Verifier.

Schnorr's Protocol

1. Prover chooses $r \in Z_q$ randomly and hands $R = g^r$ to Verifier
2. Verifier chooses $c \in Z_q$ randomly and hands it to the Prover
3. Prover computes $s = cx + r \pmod q$ and hands s to the Verifier. Verifier accepts if $X^c R = g^s$

Analysis of Schnorr's Protocol

- If Prover reuses r, then her secret x is revealed to Verifier

Analysis of Schnorr's Protocol

- If Prover reuses r, then her secret x is revealed to Verifier
- But how?

Analysis of Schnorr's Protocol

- If the Prover learns the challenge c before sending R to Verifier, Prover can successfully fake knowledge of the secret x

Analysis of Schnorr's Protocol

- If the Prover learns the challenge c before sending R to Verifier, Prover can successfully fake knowledge of the secret x
- But how?

Schnorr's Protocol

- Completeness
  - If Prover knows the secret, Verifier will be convinced of this
- Soundness
  - Prover who does not know secret x has probability $1/q$ to convince Verifier that she knows the secret
- Zero-Knowledge
  - Verifier can forge a transcript without interacting with Prover!

Schnorr Signature Scheme

Let $H : \{0,1\}^\star \rightarrow Z_q$ be a hash function.

- Setup: Signer chooses $x \in Z_q$ randomly, computes $y = g^x$ and outputs $(pk, sk) = (y, x)$.
- Signer does the following on input x and message m:
  1. chooses $r \in Z_q$ randomly and computes $R = g^r$,
  2. computes $c = H(R, m)$,
  3. computes $s = c \cdot x + r \pmod q$ and outputs signature (R, s).

Schnorr Signature Scheme

Let $H : \{0,1\}^\star \to Z_q$ be a hash function.

- Setup: Signer chooses $x \in Z_q$ randomly, computes $y = g^x$ and outputs $(pk, sk) = (y, x)$.
- Signer does the following on input x and message m:
  1. chooses $r \in Z_q$ randomly and computes $R = g^r$,
  2. computes $c = H(R, m)$,
  3. computes $s = c \cdot x + r \pmod q$ and outputs signature (R, s).
- Verifier takes the public key y, message m, and a candidate signature (R, s), and accepts iff $y^{H(R,m)} R = g^s$.

Analysis of Schnorr Signature Scheme

- Computation of c is over m. This binds proof of knowledge to a particular message m

Analysis of Schnorr Signature Scheme

- Computation of c is over m. This binds proof of knowledge to a particular message m
- Signer generates the challenge, not the Verifier! This makes the scheme non-interactive

Analysis of Schnorr Signature Scheme

- If the challenge c can be fixed without fixing R, then forgery is possible – but how?
- Suppose we define $c = H(R, m)$

Analysis of Schnorr Signature Scheme

- If the challenge c can be fixed without fixing R, then forgery is possible – but how?
- Suppose we define $c = H(R, m)$
- This ensures that c cannot be fixed without fixing R

Analysis of Schnorr Signature Scheme

- Signing the hash avoids an existential forgery attack
- To see why hash is needed, let $c = R \cdot m$

Analysis of Schnorr Signature Scheme

- Signing the hash avoids an existential forgery attack
- To see why hash is needed, let $c = R \cdot m$
- Now suppose adversary sees signature (R, s) for m
- Let $m' = 2mR^{-1}$, $s' = 2s$, $R' = R^2$

Analysis of Schnorr Signature Scheme

- Signing the hash avoids an existential forgery attack
- To see why hash is needed, let $c = R \cdot m$
- Now suppose adversary sees signature (R, s) for m
- Let $m' = 2mR^{-1}$, $s' = 2s$, $R' = R^2$
- Show that (R', s') is a valid signature for m'

Recap: EC Schnorr

- Have message m, private key k, public key $P = kG$
- Make secret nonce r, public key of nonce $R = rG$
- Challenge: $e = H(R|P|m)$
- Signature: $s = r + ek$
- Verification: $sG == R + eP$

Recap: EC Schnorr

- Have message m, private key k, public key $P = kG$
- Make secret nonce r, public key of nonce $R = rG$
- Challenge: $e = H(R|P|m)$
- Signature: $s = r + ek$
- Verification: $sG == R + eP$
- remark: Signer must first choose R before challenge e can be computed

Insecure variant

- Make secret nonce r, public key of nonce $R = rG$
- Let's say we sign a message with $e = H(P|m)$
- Signature: $s = r + ek$
- Verification: $sG == R + eP$

Insecure variant

- Make secret nonce r, public key of nonce $R = rG$
- Let's say we sign a message with $e = H(P|m)$
- Signature: $s = r + ek$
- Verification: $sG == R + eP$
- remark: Signer can fix e without calculating R
- Forgery is possible, but how?

Transformation: non-EC operations to EC operations

- Multiplication becomes point addition
- Exponentiation becomes scalar multiplication

(Exponential) ElGamal encryption scheme

- Message: m
- $(pk, sk) = (h, x)$, where $h = g^x$
- Encryption: pick a random K, compute $(c_1, c_2) = (g^K, g^m \cdot h^K)$
- Decryption:
  - Compute $g^m = c_2/(c_1^x)$
  - Solve the DLOG problem to find m (feasible if m is bounded)

(Exponential) ElGamal encryption scheme

- Message: m
- $(pk, sk) = (h, x)$, where $h = g^x$
- Encryption: pick a random K, compute $(c_1, c_2) = (g^K, g^m \cdot h^K)$
- Decryption:
  - Compute $g^m = c_2/(c_1^x)$
  - Solve the DLOG problem to find m (feasible if m is bounded)
- $E(m_1) \cdot E(m_2) = E(m_1 + m_2)$

Proof of correctness of ElGamal encryption

- Say Alice performs an ElGamal encryption of T using her public key h
- Verifier sees the ciphertext: $(c_1, c_2) = (g^K, g^T \cdot h^K)$
- Alice wants to prove that she knows the secret values (T, K) without revealing any information about them to Verifier
- But how?

ElGamal zero-knowledge proof

Prover		**Verifier**
Choose random l, m		
$r_1 = g^l$		
$r_2 = g^m \cdot h^l$	$\xrightarrow{r_1, r_2}$	
		Generate random e
	$\xleftarrow{e}$	
$s_1 = l + e \cdot K$		
$s_2 = m + e \cdot T$	$\xrightarrow{s_1, s_2}$	
		Verify
		$g^{s_1} \stackrel{?}{=} r_1 \cdot c_1^e$
		$g^{s_2} \cdot h^{s_1} \stackrel{?}{=} r_2 \cdot c_2^e$

Soundness property

- Suppose that a prover who does not know (K, T) is able to answer correctly at least two challenges e and e', with $e \neq e'$, after sending r_1, r_2.

Soundness property

- Suppose that a prover who does not know (K, T) is able to answer correctly at least two challenges e and e', with $e \neq e'$, after sending r_1, r_2.
- That is, a prover is able to produce two valid conversations $(r_1, r_2; e; s_1, s_2)$ and $(r_1, r_2; e'; s_1', s_2')$. Then it follows that the prover actually knows K and T.

Soundness property

- Suppose that a prover who does not know (K, T) is able to answer correctly at least two challenges e and e', with $e \neq e'$, after sending r_1, r_2.
- That is, a prover is able to produce two valid conversations $(r_1, r_2; e; s_1, s_2)$ and $(r_1, r_2; e'; s_1', s_2')$. Then it follows that the prover actually knows K and T.
- Therefore, after sending r_1, r_2, the prover can answer at most one challenge correctly, if the prover does not know K and T.

Proving that T is representable with n bits

- Say Alice performs an ElGamal encryption of a value T using her public key h
- Alice wants to prove that T can be represented using n or fewer bits, implying $T \leq 2^n - 1$
- But how?

Proving that T is representable with n bits

- Write T in its binary representation:

$$T = \sum_{i=0}^{n-1} 2^i \cdot b_i, \text{ where } b_i \in \{0, 1\}$$

- Encrypt each bit b_i using the public key h:

$$(c_{1i}, c_{2i}) = (g^{K_i}, g^{b_i} \cdot h^{K_i}), 0 \leq i \leq n - 1$$

- But how can Verifier check correctness of the above ciphertexts?

Proving that T is representable with n bits

- Write T in its binary representation:

$$T = \sum_{i=0}^{n-1} 2^i \cdot b_i, \text{ where } b_i \in \{0, 1\}$$

- Encrypt each bit b_i using the public key h:

$$(c_{1i}, c_{2i}) = (g^{K_i}, g^{b_i} \cdot h^{K_i}), 0 \leq i \leq n-1$$

- But how can Verifier check correctness of the above ciphertexts?
- Execute an OR-proof for every encrypted bit so that verifier is convinced that it is the encryption of 0 or 1, but does not learn any additional information besides that

Proof of the encryption of 0

- put $m = 0$ and $T = 0$ in our previous protocol

Prover		**Verifier**
Choose random l		
$r_1 = g^l$		
$r_2 = h^l$	$\xrightarrow{r_1, r_2}$	
		Generate random e
	$\xleftarrow{e}$	
$s = l + e \cdot K$	$\xrightarrow{s}$	
		Verify
		$g^s \stackrel{?}{=} r_1 \cdot c_1^e$
		$h^s \stackrel{?}{=} r_2 \cdot c_2^e$

Proof of the encryption of 1

- put $m = 1$ and $T = 1$ in our previous protocol

Prover		**Verifier**
Choose random l, m		
$r_1 = g^l$		
$r_2 = g \cdot h^l$	$\xrightarrow{r_1, r_2}$	
		Generate random e
	$\xleftarrow{e}$	
$s = l + e \cdot K$	$\xrightarrow{s}$	
		Verify
		$g^s \stackrel{?}{=} r_1 \cdot c_1^e$
		$g^{e+1} \cdot h^s \stackrel{?}{=} r_2 \cdot c_2^e$

Proving that the bit representation represents T

- Verifier has the ciphertext: $(c_1, c_2) = (g^K, g^T \cdot h^K)$
- Both the Verifier and Alice compute the following ciphertext:

$$
\begin{aligned}
\tilde{c} &= \left(\frac{c_1}{\prod_{i=0}^{n-1} c_{1i}^{2^i}}, \frac{c_2}{\prod_{i=0}^{n-1} c_{2i}^{2^i}} \right) \\
&= \\
&= \left(g^{K - \sum_{i=0}^{n-1} 2^i \cdot K_i}, g^{T - \sum_{i=0}^{n-1} 2^i \cdot b_i} \cdot h^{K - \sum_{i=0}^{n-1} 2^i \cdot K_i} \right)
\end{aligned}
$$

Proving that the bit representation represents T

- Verifier has the ciphertext: $(c_1, c_2) = (g^K, g^T \cdot h^K)$
- Both the Verifier and Alice compute the following ciphertext:

$$
\begin{aligned}
\tilde{c} &= \left(\frac{c_1}{\prod_{i=0}^{n-1} c_{1i}^{2^i}}, \frac{c_2}{\prod_{i=0}^{n-1} c_{2i}^{2^i}} \right) \\
&= \\
&= \left(g^{K - \sum_{i=0}^{n-1} 2^i \cdot K_i}, g^{T - \sum_{i=0}^{n-1} 2^i \cdot b_i} \cdot h^{K - \sum_{i=0}^{n-1} 2^i \cdot K_i} \right)
\end{aligned}
$$

- Now Alice proves that $\tilde{c} = (g^{\tilde{K}}, h^{\tilde{K}})$ is an encryption of 0
  - This proof is exactly the same as our previous protocol for the proof of the encryption of 0

Proving that the bit representation represents T

- Verifier has the ciphertext: $(c_1, c_2) = (g^K, g^T \cdot h^K)$
- Both the Verifier and Alice compute the following ciphertext:

$$
\begin{aligned}
\tilde{c} &= \left(\frac{c_1}{\prod_{i=0}^{n-1} c_{1i}^{2^i}}, \frac{c_2}{\prod_{i=0}^{n-1} c_{2i}^{2^i}} \right) \\
&= \\
&= \left(g^{K-\sum_{i=0}^{n-1} 2^i \cdot K_i}, g^{T-\sum_{i=0}^{n-1} 2^i \cdot b_i} \cdot h^{K-\sum_{i=0}^{n-1} 2^i \cdot K_i} \right)
\end{aligned}
$$

- Now Alice proves that $\tilde{c} = (g^{\tilde{K}}, h^{\tilde{K}})$ is an encryption of 0
  - This proof is exactly the same as our previous protocol for the proof of the encryption of 0
- This proves that the bit representation actually represents T