Lecture 10: NIZKs

4/29/21

Plan

Sigma protocols for XOR and AND gates

NIZKs
- What are NIZKs?
- Sigma protocols $\rightarrow$ NIZKs

Schnorr proofs revisited
- Recall protocol
- identification scheme
- Signatures

Example: private polling application

Proofs of AND/XOR of committed bits

Let (G, g, h) be params for Pedersen commitment.

Suppose we have 3 commitments c_1, c_2, c_3

Prover wants to convince Verifier that it knows

$m_1, m_2, m_3 \in \{0,1\}$, $r_1, r_2, r_3 \in \mathbb{Z}_q$

s.t. $\forall i \in \{1,2,3\}$ $c_i = g^{m_i} h^{r_i}$ and $m_1 \wedge m_2 = m_3$

This corresponds to $\mathcal{L}_{AND}$ that you are given in HW3 problem 3.

Idea: Since m_1, m_2, m_3 are bits, there are only 8 possible combos, only 4 of which are in the language $\mathcal{L}_{AND}$.

So it suffices to prove

$(m_1 = 0$ AND $m_2 = 0$ AND $m_3 = 0)$ OR

$(m_1 = 0$ AND $m_2 = 1$ AND $m_3 = 0)$ OR

$(m_1 = 1$ AND $m_2 = 0$ AND $m_3 = 0)$ OR

$(m_1 = 1$ AND $m_2 = 1$ AND $m_3 = 1)$

We know how to do AND/OR of Sigma protocols from last time, so we just need to see how to prove that c_i commits to 0 or 1.

To prove $m=0$: $c = g^0 h^r = h^r$

prove knowledge of $\log_h c$ via Schnorr proof

To prove $m=1$: $c = g^1 h^r \Rightarrow h^r = c \cdot g^{-1}$

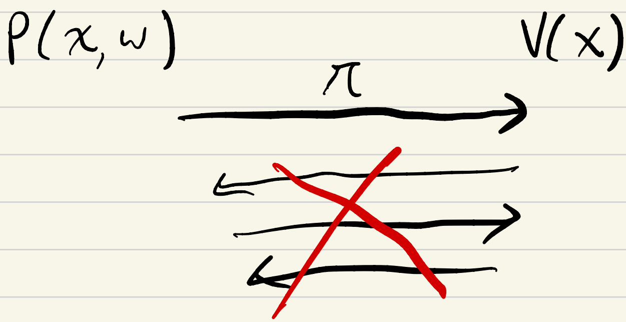
prove knowledge of $\log_h (c \cdot g^{-1})$

NIZKs

Sigma protocols are nice because they give us 3-message ZK protocols.

Can we do even better? Can we get 1-message ZK protocols?

A.K.A. Non-Interactive Zero Knowledge (NIZK)



Can we have ZK "conventional" proofs?

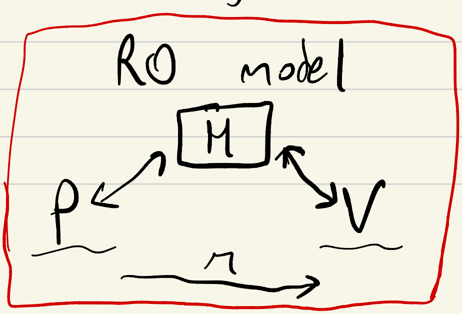
Perhaps unsurprisingly, it turns out such proofs only exist for "easy" languages ($\mathcal{L} \in BPP$) in the standard model.

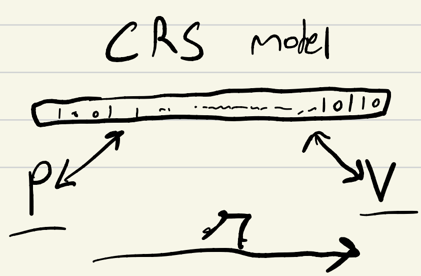
Bounded-error Probabilistic Polynomial time (like class P + access to randomness)

Why? Intuitively, when proof is 1 message, Sim alg should be able to output the message.

But we can get NIZKs if we change the model!!

very surprising





Fiat-Shamir

Convert Sigma protocol for language $\mathcal{L}$ $\rightarrow$ NIZKPoK for $\mathcal{L}$ in RO model

Sigma protocols:

$\underline{P((x,w)\in R)}$ $\qquad\qquad\qquad$ $\underline{V(x)}$

$\xrightarrow{\text{commitment } t}$

$\xleftarrow{\text{challenge } c}$ $\qquad c \xleftarrow{R} C$

challenge chosen uniformly at random

$\xrightarrow{\text{response } z}$

$\downarrow$

acc/rej as deterministic function of (x,t,c,z)

1) Completeness
2) Special soundness
3) Special HVZK

Notice that V 1) sends only random values to P
$\qquad\qquad\qquad$ 2) has no secret state

$\Big\}$ we call this "public coin"

Fiat-Shamir idea: Replace the Verifier's message with with the random oracle!
$c \leftarrow H(x,t) \in \mathbb{Z}_q$

$P((x,w)\in R)$ $\qquad\qquad\qquad$ $V(x)$

Prove, set $c \leftarrow H(x,t)$ $\qquad \xrightarrow{\pi = t, z}$ $\qquad$ recompute $c \leftarrow H(x,t)$

$\downarrow$

acc/rej as deterministic function of (x,t,c,z)

What properties do we need to prove?

Completeness → follows directly from completeness of Sigma protocol

ZK → follows from HVZK of underlying Sigma protocol,
Sim programs RO with choice of c.

This is why we focused on HVZK for Sigma protocols
the RO behaves like an honest verifier.

Soundness/Knowledge → Ext behaves just like Sigma protocol extractor,
except instead of rewinding & sending new challenge, rewind and
reprogram random oracle for new challenge.

("Special" soundness/HVZK properties mentioned last time are sufficient to make this work formally)

More Schnorr

Recall protocol:

$$
\begin{array}{lcl}
\underline{P(x \in \mathbb{Z}_q,\ h = g^x \in G)} & & \underline{V(h \in G)} \\
r \xleftarrow{R} \mathbb{Z}_q & \xrightarrow{\ u = g^r\ } & \\
 & \xleftarrow{\ c\ } & c \xleftarrow{R} \mathbb{Z}_q \\
 & \xrightarrow{\ z = r + cx\ } & \text{check} \\
 & & g^z = u \cdot h^c
\end{array}
$$

prover is showing that it can come up with another representation of $u \cdot h^c$ b/c it knows dlog of h.

An immediate application of Schnorr's protocol: identification protocol

Goal: client wants to authenticate to server s.t. eavesdropping adversary can't steal login credential.

$$
\begin{array}{lcl}
\underline{\text{Client}} & & \underline{\text{Server}} \\
\text{holds secret } x & \longrightarrow & \text{holds verification key } h = g^x \\
 & \longleftarrow & \\
 & \longrightarrow & \\
 & \text{run Schnorr protocol} &
\end{array}
$$

PoK → only client who knows secret can authenticate

HVZK → eavesdropper learns nothing about secret from transcript

Schnorr Signatures

Using Schnorr protocol + Fiat-Shamir can actually give us a signature!

Let's see what Schnorr + Fiat-Shamir looks like:

$P(x \in \mathbb{Z}_q,\ h = g^x \in G)$		$V(h \in G)$
$r \xleftarrow{R} \mathbb{Z}_q$	$\xrightarrow{\quad u = g^r \quad}$	$\cancel{c \xleftarrow{R} \mathbb{Z}_q}$
	$c \leftarrow H(h, u)$ ~~$\longleftarrow$~~	
	$\xrightarrow{\quad z = r + cx \quad}$	check $c = H(h, u)$
		check $g^z = u \cdot h^c$

How to make this a signature? Add m as an input to hash

$$c \leftarrow H(h, u, m)$$

Intuitively, this works because forging a signature requires a proof of knowledge of the secret, and seeing signatures on other messages doesn't help either because of the ZK property of the NIZK. See book for actual proof.

Schnorr Signatures

in a group G of prime order q with generator g where dlog hard:

KeyGen() $\quad x \xleftarrow{R} \mathbb{Z}_q$

$\quad sk \leftarrow x, \; pk \leftarrow g^x \in G \qquad$ pick x, $h = g^x$

Sign(sk, m) $\quad r \xleftarrow{R} \mathbb{Z}_q \quad R = g^r \in G \qquad$ first prover message

$\quad c \leftarrow H(pk, R, m) \qquad$ verifier message

$\quad z \leftarrow r + cx \in \mathbb{Z}_q \qquad$ second prover message

$\quad$ output $(R, c, z) \qquad$ protocol messages

Verify($pk^{=g^x}$, (R, c, z), m) $\quad$ check $c = H(pk, R, m)$

$\quad$ check $g^z = R \cdot pk^c \qquad$ role of verifier

practical notes:
- in this specific case, don't need pk in hash
- can omit R from sig since verify can recompute it as $R = g^z \cdot (pk^c)^{-1}$ and then check that c was computed correctly
- Soundness error is $1/|C|$, so c can be 128 bits
- z is in $\mathbb{Z}_q$, which would be 256 bits for EC gp.

So total size of sig can be $256 + 128 = 384$ bits

By comparison, RSA-FDH sig is 3072 bits

BLS signature is ~~256 bits~~ $\rightarrow$ 384 bits for comparable security

In practice, ECDSA signatures are widely used

Same idea as Schnorr, but worse. Why? patents! (expired 2008)

Example: a private polling application

Suppose we want a privacy-preserving poll on whether or not to add an extra HW assignment.

Student 1 $\xrightarrow{\text{Vote}}$
Student 2 $\xrightarrow{\text{Vote}}$ Tallying center (course staff) $\longrightarrow \sum_i b_i$
⋮ $\xrightarrow{\text{Vote}}$
$b_i \in \{0,1\}$ Student N $\xrightarrow{\text{Vote}}$

Warning: This looks like an election, but is much simpler.

How do we do this such that an honest course staff doesn't learn the individual votes?

Idea 1: use Additively homomorphic public key encryption so course staff can add up all the votes before decrypting!

We saw that Pedersen commitments are additively homomorphic in lecture 3, but do we know an additively homomorphic encryption?

Yes! El-Gamal encryption (from CS 255)

KeyGen(1^λ) $\quad x \xleftarrow{R} \mathbb{Z}_q$
$\quad pk \leftarrow g^x$

Enc(pk, m) $\quad r \xleftarrow{R} \mathbb{Z}_q$
$\quad U \leftarrow g^r$
$\quad V \leftarrow pk^r g^m$
$\quad$ output (U,V)

We'll use $m \in \{0,1\}$

Dec$(sk, (U,V))$ Output $V \cdot (U^{sk})^{-1}$

Correctness: $V \cdot (U^{sk})^{-1} = pk^r g^m / (g^r)^{sk}$
$= g^{xr+m} / g^{xr}$
$= g^{xr+m-xr} = g^m$

El-Gamal Encryption is additively homomorphic (for small msg space)

to add ciphertexts (U_1, V_1), (U_2, V_2)

$$U = U_1 \cdot U_2 = g^{r_1} \cdot g^{r_2} = g^{r_1+r_2}$$

$$V = V_1 \cdot V_2 = pk^{r_1} g^{m_1} \cdot pk^{r_2} g^{m_2} = pk^{(r_1+r_2)} g^{(m_1+m_2)}$$

As long as msg space is small, easy to recover m_1+m_2 from $g^{m_1+m_2}$

So we use El-Gamal encryption to encrypt votes and send them to the course staff, who sum and decrypt. Done? Nope. What if a malicious student really wants more HW? Instead of choosing $b=1$, the student could pick $b=100$ and "stuff the ballot box" with 100 votes, overwhelming the preferences of the rest of the class.

Solution: Each student encrypts their vote and gives a non-interactive Zero Knowledge proof of Knowledge that they have encrypted either 0 or 1. The tallying center verifies each proof before adding the corresponding encryption to the sum.

We saw earlier in this lecture how to prove that a commitment is to 0 or 1. The approach for an El-Gamal ciphertext will be similar, but it requires a slightly different proof system. Instead of a Schnorr proof, we will use a Chaum-Pedersen proof.

Chaum-Pedersen is a proof that a given triple is a DDH triple, i.e. given public $w, u, v \in G$, I know x s.t. $u = g^x$, $v = w^x$

This implies for $w = g^y$ that $(w, u, v) = g^y, g^x, g^{yx}$

$P(x, (w,u,v))$		$V(w,u,v)$
$r \xleftarrow{R} \mathbb{Z}_q$		
$u' \leftarrow g^r$		
$v' \leftarrow w^r$	$\xrightarrow{u', v'}$	
	$\xleftarrow{c}$	$c \xleftarrow{R} C$
$z = r + xc$		
	$\xrightarrow{z}$	check
		$g^z = u' \cdot u^c$
		$w^z = v' \cdot v^c$

like 2 schnorr pfs w/same chal/response. Can apply Fiat-Shamir as before.

How to use to prove el-gamal encryption encrypts 0?

PK, u, v from elgamal is DDH tuple $(\overset{PK}{g^x}, g^r, (g^x)^r \cdot g^0)$

How to use to prove el-gamal encryption encrypts 1?

PK, u, $v \cdot g^{-1}$ is DDH tuple $(g^x, g^r, (g^x)^r \cdot \cancel{g^1 g^{-1}})$