Lecture 10: NIZKs

5/7/20
Plan

Sigma protocols for XOR and AND gates

NIZKs
  - What are NIZKs?
  - Sigma protocols → NIZKs

Schnorr proofs revisited
  - Recall protocol
  - Identification scheme
  - Signatures

Example: private polling application
Proofs of AND/XOR of committed bits

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

Suppose we have 3 commitments \(c_1, c_2, c_3\).

Prover wants to convince Verifier that it knows

\[ m_1, m_2, m_3 \in \{0,1\}, \quad r_1, r_2, r_3 \in \mathbb{Z}_p \]

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

This corresponds to \( L_{\text{AND}} \) 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 \( L_{\text{AND}} \).

So it suffices to prove

\[
( m_1 = 0 \text{ AND } m_2 = 0 \text{ AND } m_3 = 0 ) \text{ OR }
( m_1 = 0 \text{ AND } m_2 = 1 \text{ AND } m_3 = 0 ) \text{ OR }
( m_1 = 1 \text{ AND } m_2 = 0 \text{ AND } m_3 = 0 ) \text{ OR }
( m_1 = 1 \text{ AND } m_2 = 1 \text{ AND } m_3 = 1 )
\]

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?

\[
P(x, w) \xrightarrow{\pi} V(x)
\]

\(\n\]

AHA. Non-Interactive Zero Knowledge (NIZK)

\[P(x, w) \xrightarrow{\pi} V(x)\]

Can we have ZK "conventional" proofs?

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

\[P(x, w) \xrightarrow{\pi} V(x)\]

Bounded-error Probabilistic Polynomial time (like class \(P + \text{access to randomness}\)).

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!!

\[P(x, w) \xrightarrow{\pi} V(x)\]

\[\text{RO model}\]

\[\text{CRS model}\]

\[p \xrightarrow{\pi} V\]
Fiat-Shamir

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

Sigma protocols:

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

commit $t$ → $V(x)$

challenge $c$

response $z$

1) Completeness
2) Special soundness
3) Special HVZK

Notice that $V 1)$ sends only random values to $P$, we call this "public coin"
2) has no secret state

Fiat-Shamir idea: Replace the Verifier's message with the random oracle!

$c \leftarrow H(x, t) \in \mathbb{Z}_q$

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

prove, set

$z = t, z$

$V(x)$

$\rightarrow$

recompute $c \leftarrow H(x, t)$

accept/reject as deterministic function of $(x, t, z)$
What properties do we need to prove?

Completeness \(\rightarrow\) follows directly from completeness of Sigma protocol

ZK \(\rightarrow\) follows from HV2K of underlying Sigma protocol, Sim programs RO with choice of C.

This is why we focused on HV2K for sigma protocols the RO behaves like an honest verifier.

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

("Special" soundness/HV2K properties mentioned last time are sufficient to make this work formally)
More Schnorr

Recall protocol:

\[ P(x \in \mathbb{Z}_p, h=g^x \in G) \]

\[ r \in \mathbb{Z}_p \]

\[ U=g^r \]

\[ L \]

\[ Z = r + c \cdot x \]

\[ c \in \mathbb{Z}_p \]

\[ check \]

\[ g^Z = U \cdot h^c \]

Prover is showing that it can come up with another representation of \( U \cdot h^c \) \( \text{b/c } \) it knows \( \log h \).

An immediate application of Schnorr's protocol: identification protocol

**Goal:** Client wants to authenticate to server \( \text{s.t. eavesdropping adversary can't steal login credential.} \)

Client

\[ \text{holds secret } x \]

\[ \rightarrow \]

Server

\[ \text{holds verification key } h=g^x \]

\[ \leftarrow \]

Run Schnorr protocol

Prover \( \rightarrow \) only client who knows secret can authenticate

HV2H \( \rightarrow \) 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:

\[ P(x \in \mathbb{Z}_p, h = g^x \in G) \]

\[ r \in \mathbb{Z}_p \]

\[ u = g^r \]

\[ c = H(h, u) \]

\[ z = r + cx \]

\[ V(h \in G) \]

\[ c \in \mathbb{Z}_p \]

Check: \[ c = H(h, u) \]

\[ g^z = u \cdot h^c \]

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 $g$ whose log hard:

$$\text{KeyGen}(): \ x \leftarrow \mathbb{Z}_q$$

$$s_k \leftarrow x, \ \text{pk} \leftarrow g^x \in G \quad \text{pck} \ x, \ h = g^x$$

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

$$c \leftarrow H(pk, R, m) \quad \text{verifier message}$$

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

$$\text{output} (R, C, z) \quad \text{protocol messages}$$

$$\text{Verify}(pk, (R, C, z), m): \ \text{check} \quad c = H(pk, R, m) \quad \text{role of verifier}$$

$$\text{check} \quad g^z = R \cdot pk^c$$

practical notes:
- in this specific case, don't need pck in pck
- can omit $R$ from sig since verify can recompute it as $R = g^z \cdot (pk)^{-1}$ and then check that $c$
  was computed correctly
- soundness error is $\sqrt{\text{cl}}$, 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

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.

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)

\[
\begin{align*}
\text{KeyGen}(1^n) & : x \in \mathbb{Z}_q \\
& \quad \text{pk} \leftarrow g^x \\
& \quad \text{sk} \\
\text{Enc}(\text{pk}, m) & : r \in \mathbb{Z}_q \\
& \quad U \leftarrow g^r \\
& \quad V \leftarrow \text{pk}^r g^m \\
\text{Dec}(\text{sk}, (u,v)) & : \text{output } V \cdot (u^s)^{-1} \\
\text{Correctness: } V \cdot (u^s)^{-1} & = \text{pk}^r g^m / (g^r)^s \\
& = g^{x r + n} / g^{x r} \\
& = g^{x r + m - x r} = g^m
\end{align*}
\]
El-Gamal Encryption is additively homomorphic (for small msg space)

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

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

\[
V = V_1 \cdot U_2 = \text{pk}^{r_1} g^{r_1} \cdot \text{pk}^{r_2} g^{r_2} = \text{pk}^{(r_1+r_2)} g^{r_1+r_2}
\]

As long as msg space is small, easy to recover \(r_1+r_2\) from \(g^{r_1+r_2}\)

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 \( u, u, v \in G \), I know \( x \) s.t. \( U = g^x, V = w^x \).

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

\[
\begin{align*}
\Pr(x, (u, u, v)) & \quad \Pr(v, (w, u, v)) \\
r & \in \mathbb{Z}_q \\
u & \in g^r \\
v & \in W^r \\
v' & \in \mathbb{G} \\
v' & \in \mathbb{G} \\
U & \in \mathbb{G} \\
U & \in \mathbb{G} \\
C & \in \mathbb{G} \\
C & \in \mathbb{G} \\
Z & = r + xc \\
Z & = r + xc \\
g^z & = U^x . U^c \\
w^z & = V^x . V^c \\
\text{like 2 schnorr pfs w/same chal/response. Can apply Fiat-Shamir as before.}
\end{align*}
\]

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

\( PK, u, v \) from elgamal is DDH tuple \( (g^x, g^r, (g^x)^v, g^0) \)

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

\( PK, u, v, g^t \) is DDH tuple \( (g^x, g^r, (g^x)^v, g^t, g^t^*) \)