Advanced Topics in Cryptography

Lecture 3

Benny Pinkas

Based on slides of Yehuda Lindell

Sigma Protocol for proving a DH Tuple

- Relation R of Diffie-Hellman tuples
  - $(g,h,u,v) \in R$ iff there exists w s.t. $u=g^w$ and $v = h^w$
  - Useful in many protocols
- This is a proof of membership, not of knowledge

- Protocol
  - **P** chooses a random r and sends $a=g^r$, $b=h^r$
  - **V** sends a random e
  - **P** sends $z=r+ew$ mod q
  - **V** checks that $g^z=au^e$, $h^z=bv^e$

Sigma Protocol DH Tuple

- Completeness: as in DLOG
- Special soundness:
  - Given $(a,b,e,z),(a,b,e',z')$, we have $g^z=au^e, g^{z'}=au^{e'}, h^z=bv^e, h^{z'}=bv^{e'}$ and so like in DLOG on both
    - $w = (z-z')/(e-e')$
- Special HVZK
  - Given (g,h,u,v) and e, choose random z and compute
    - $a = g^z u^{-e}$
    - $b = h^z v^{-e}$

P $((g,h,u,v),w)$ **V**

$$\xrightarrow{a=g^r,\ b=h^r}$$

$$\xleftarrow{e}$$

$$\xrightarrow{z=r+ew}$$

$$g^z \stackrel{?}{=} au^e$$
$$h^z = bv^e$$

Tools for Sigma Protocols

- Prove compound statements
  - AND, OR, subset
- ZK from sigma protocols
  - Can first make a compound sigma protocol and then compile it
- ZKPOK from sigma protocols

AND of Sigma Protocols

- To prove the AND of multiple statements
  - Run all in parallel
  - Can use the same verifier challenge **e** in all

- Sometimes it is possible to do better than this
  - Statements can be batched
  - E.g. proving that many tuples are DDH can be done in much less time than running all proofs independently
    - Batch all into one tuple and prove

OR of Sigma Protocols

- This is more complicated
  - Given two statements and two appropriate Sigma protocols, wish to prove that at least one is true, without revealing which

- The solution – an ingenious idea from [CDS]
  - Using the simulator, if **e** is known ahead of time it is possible to cheat
  - We construct a protocol where the prover can cheat in one out of the two proofs

OR of Sigma Protocols

- The template for proving x_0 or x_1:
  - **P** sends two first messages $(\mathbf{a_0},\mathbf{a_1})$
  - **V** sends a single challenge $\mathbf{e}$
  - **P** replies with
    - Two challenges $\mathbf{e_0},\mathbf{e_1}$ s.t. $\mathbf{e_0} \oplus \mathbf{e_1} = \mathbf{e}$
    - Two final messages $\mathbf{z_0},\mathbf{z_1}$
  - **V** accepts if $\mathbf{e_0} \oplus \mathbf{e_1} = \mathbf{e}$ and $(\mathbf{a_0},\mathbf{e_0},\mathbf{z_0}),(\mathbf{a_1},\mathbf{e_1},\mathbf{z_1})$ are both accepting
- How does this work?

OR of Sigma Protocols

- **P** sends two first messages $(\mathbf{a_0},\mathbf{a_1})$
  - Suppose that **P** has a witness for $\mathbf{x_0}$ (but not for $\mathbf{x_1}$)
  - **P** chooses a random $\mathbf{e_1}$ and runs SIM to get $(\mathbf{a_1},\mathbf{e_1},\mathbf{z_1})$
  - **P** sends $(\mathbf{a_0},\mathbf{a_1})$
- **V** sends a single challenge **e**
- **P** replies with e_0,e_1 s.t. $e_0 \oplus e_1 = e$ and with z_0,z_1
  - **P** already has $\mathbf{z_1}$ and can compute $\mathbf{z_0}$ using the witness
- Soundness
  - If P doesn't know a witness for $\mathbf{x_1}$, he can only answer for a single $\mathbf{e_1}$
  - This means that **e** defines a single challenge $\mathbf{e_0}$, like in a regular proof

OR of Many Statements

- Prove k out of n statements $x_1,\ldots,x_n$
  - **A** = set of indices that prover knows how to prove; the other indices are denoted as **B.** $|A|=k$. $|B|=n-k$.
  - Use secret sharing with threshold $n-k+1$
  - Field elements $1,2,\ldots,n$. Polynomial **f** of degree $n-k$
  - Share for party $\mathbf{P}_i$: $\mathbf{f}(i)$
- Prover
  - For every $\mathbf{i}\in\mathbf{B}$, prover generates $(\mathbf{a}_i,\mathbf{e}_i,\mathbf{z}_i)$ using SIM
  - For every $\mathbf{j}\in\mathbf{A}$, prover generates $\mathbf{a}_j$ as in protocol
  - Prover sends $(\mathbf{a}_1,\ldots,\mathbf{a}_n)$

OR of Many Statements

- Prover sent $(\mathbf{a_1,\ldots,a_n})$
- Verifier sends a random field element $e \in F$
- Prover finds the (only) polynomial f of degree n-k passing through all (i,e_i) and $(0,e)$ (for $i \in B$)
  - The prover computes $\mathbf{e_j=f}(j)$ for every $\mathbf{j \in A}$
  - The prover computes $\mathbf{z_j}$ as in the protocol, based on transcript $\mathbf{a_j,e_j}$
- Soundness follows because there are |F| possible vectors and the prover can only answer one

General Compound Statements

- This can be generalized to any monotone formula (meaning that the formula contains AND/OR but no negations)
  - See Cramer, Damgård, Schoenmakers, Proofs of partial knowledge and simplified design of witness hiding protocols, CRYPTO'94.

ZK from Sigma Protocols

- A tool: commitment schemes
- Enables to commit to a chosen value while keeping it secret, with the ability to reveal the committed value later.
- A commitment has two properties:
  - Binding: After sending the commitment, it is impossible for the committing party to change the committed value.
  - Hiding: By observing the commitment, it is impossible to learn what is the committed value. (Therefore the commitment process must be probabilistic.)
- It is possible to have unconditional security for any one of these properties, but not for both.

ZK from Sigma Protocols

- The basic idea
  - Have **V** first commit to its challenge **e** using a perfectly-hiding commitment
- The protocol
  - **P** sends the first message α of the commit protocol
  - **V** sends a commitment c=$\mathbf{Com}_\alpha(\mathbf{e};\mathbf{r})$
  - **P** sends a message **a**
  - **V** opens the commitment by sending (**e**,**r**)
  - **P** checks that c=$\mathbf{Com}_\alpha(\mathbf{e};\mathbf{r})$ and if yes sends a reply **z**
  - **V** accepts based on (**x**,**a**,**e**,**z**)

ZK from Sigma Protocols

- Soundness:
  - The perfectly hiding commitment reveals nothing about **e** and so soundness is preserved

- Zero knowledge
  - In order to simulate:
    - Send **a′** generated by the simulator, for a random **e′**
    - Receive **V**'s decommitment to **e**
    - Run the simulator again with **e**, rewind **V** and send **a**
      - Repeat until **V** decommits to **e** again
    - Conclude by sending **z**
  - Analysis…

Pedersen Commitments

- Highly efficient perfectly-hiding commitments (two exponentiations for string commit)
  - **Parameters:** generator **g**, order **q**
  - **Commit protocol** (commit to **x**):
    - Receiver chooses random **k** and sends $h=g^k$
    - Sender sends $c=g^r h^x$, for random **r**
  - **Hiding:**
    - For every **x,y** there exist **r,s** s.t. $r+kx = s+ky \bmod q$
  - **Binding:**
    - If sender can open commitment in two ways, i.e. find $(x,r),(y,s)$ s.t. $g^r h^x=g^s h^y$, then $k = (r-s)/(y-x) \bmod q$

Efficiency of ZK

- Using Pedersen commitments, the entire DLOG proof costs only 5 additional group exponentiations
  - In Elliptic curve groups this is very little

ZKPOK from Sigma Protocols

- Is the previous protocol a proof of knowledge?
  - It seems not to be
  - The extractor for the Sigma protocol needs to obtain two transcripts with the same **a** and different **e**
    - The prover may choose its first message **a** differently for every commitment string.
    - But in this protocol the prover sees a commitment to **e** before sending **a**.
    - So if the extractor changes **e**, the prover changes **a**

ZKPOK from Sigma Protocols

- Solution: use a trapdoor (equivocal) commitment scheme
  - Given a trapdoor, it is possible to open the commitment to any value
- Pedersen has this property – given the discrete log k of **h**, can decommit to any value
  - Commit to **x**: $\mathbf{c = g^r h^x}$
  - To decommit to **y**, find **s** such that $\mathbf{r+kx = s+ky}$
  - This is easy if **k** is known: compute $\mathbf{s = r+k(x-y) \bmod q}$

ZKPOK from Sigma Protocols

- The basic idea
  - Have **V** first commit to its challenge **e** using a perfectly-hiding trapdoor (equivocal) commitment
- The protocol
  - **P** sends the first message α of the commit protocol (e.g., including h in the case of Pedersen commitments).
  - **V** sends a commitment c=$\mathbf{Com}_{\alpha}(\mathbf{e};\mathbf{r})$
  - **P** sends a message **a**
  - **V** sends (**e**,**r**)
  - **P** checks that c=$\mathbf{Com}_{\alpha}(\mathbf{e};\mathbf{r})$ and if yes sends the **trapdoor** for the commitment and **z**
  - **V** accepts if the **trapdoor** is correct and (**x,a,e,z**) is accepting

ZKPOK from Sigma Protocols

P (x,w) **V** (x)

$h = g^k$, random **k** →

← $c = g^r h^e$

Sigma msg **a** →

← **(e,r)**

Verify $c = g^r h^e$

(k,z) →

Verify $h = g^k$

Verify (a,e,z)

ZKPOK from Sigma Protocols

- Why does this help?
  - **Zero-knowledge** remains the same
  - **Extraction:** after verifying the proof once, the extractor obtains **k** and can rewind back to the decommitment of **c** and send any (**e′**,**r′**)

- Efficiency:
  - Just 6 exponentiations (very little)

ZK and Sigma Protocols

- We typically want zero knowledge, so why bother with sigma protocols?
  - There are many useful general transformations
    - E.g., parallel composition, compound statements
    - The ZK and ZKPOK transformations can be applied on top of the above, so obtain transformed ZK
  - It is **much harder** to prove ZK than Sigma
    - ZK – distributions and simulation
    - Sigma: only HVZK and special soundness

Using Sigma Protocols and ZK

- Prove that the El Gamal encryption (u,v) under public-key (g,h) is to the value m
  - By the definition of El Gamal encryption: $\mathbf{u=g^r}$, $\mathbf{v=h^r \cdot m}$
  - Thus (**g,h,u,v/m**) is a DH tuple
  - So, given (**g,h,u,v,m**), just prove that (**g,h,u,v/m**) is a DH tuple

Another application: Efficient Coin Tossing

- P_1 chooses a random x, sends $(g,h,g^r,h^r x)$
- P_1 ZK-proves that it knows the encrypted value
  - Suffices to prove that it knows the discrete log of **h**
- P_2 chooses a random y and sends to P_1
- P_1 sends x (without decommitting)
- P_1 ZK-proves that encrypted value was x
- Both parties output x+y

- Cost: O(1) exponentiations

Prove Knowledge of Committed Value

- Relation: $((h,c),(x,r)) \in R$ iff $c = g^r h^x$
- Sigma protocol:
  - P chooses random α, β and sends $\mathbf{a = h^{\alpha} g^{\beta}}$
  - V sends a random $\mathbf{e}$
  - P sends $\mathbf{u = \alpha + ex}$, $\mathbf{v = \beta + er}$
  - V checks that $\mathbf{h^u g^v = ac^e}$
- Completeness:
  - $\mathbf{h^u g^v = h^{\alpha+ex} g^{\beta+er} = h^{\alpha} g^{\beta} (h^x g^r)^e = ac^e}$

Pedersen Commitment Proof

- Special soundness:
  - Given (**a,e,u,v**),(**a,e′,u′,v′**), we have $h^u g^v = a c^e$, $h^{u'} g^{v'} = a c^{e'}$

    Thus, $h^u g^v c^{-e} = h^{u'} g^{v'} c^{-e'}$

    and $h^{u-u'} g^{v-v'} = c^{e-e'}$
  - Conclude: $x = (u-u')(e-e')$ and
    $r = (v-v')(e-e')$

P ((h,c),(x,r)) **V**

$a = h^{\alpha} g^{\beta}$ →

← e

$u = \alpha + ex$, $v = \beta + er$ →

$h^u g^v \stackrel{?}{=} a c^e$

- Special HVZK
  - Given (**g,h,h,c**) and **e**, choose random **u,v** and compute $a = h^u g^v c^{-e}$

Proof of Pedersen Value

- Prove that the Pedersen committed value is x
- Relation: $((h,c,x),(r)) \in R$ iff $c=g^r h^x$
  - Observe: $\mathbf{ch^{-x} = g^r}$
  - Conclusion: just prove that you know the discrete log of $\mathbf{ch^{-x}}$

- Application: statistical coin tossing