Sigma Protocols

Benny Pinkas

Bar-Ilan University

Zero Knowledge

- Prover P, verifier V, language L
- P proves that $x \in L$ without revealing anything
  - **Completeness: V** always accepts when honest **P** and **V** interact
  - **Soundness: V** accepts with negligible prob when **x**$\notin$**L**, for any **P***
    - Computational soundness: only holds when **P*** is polynomial-time
  - **Zero-knowledge:** There exists a simulator **S** such that **S**(**x**) is indistinguishable from a real proof execution

ZK Proof of Knowledge

- Prover P, verifier V, relation R
- P proves that it **knows** a witness w for which $(x,w) \in R$ without revealing anything

- How can one prove that is “knows” something?
- The approach used: A machine knows something if the machine can be used to efficiently compute it.

ZK Proof of Knowledge

- Prover P, verifier V, relation R
- P proves that it **knows** a witness w for which $(x,w) \in R$ without revealing anything
  - There exists an extractor **K** that can obtain from P a witness **w** such that $(\mathbf{x},\mathbf{w}) \in \mathbf{R}$ (succeeds with the same prob that $\mathbf{P}^*$ convinces **V**)

- Equivalently: The protocol securely computes the functionality $\mathbf{f}_{\mathbf{zk}}((\mathbf{x},\mathbf{w}),\mathbf{x}) = (-,\mathbf{R}(\mathbf{x},\mathbf{w}))$

Zero Knowledge

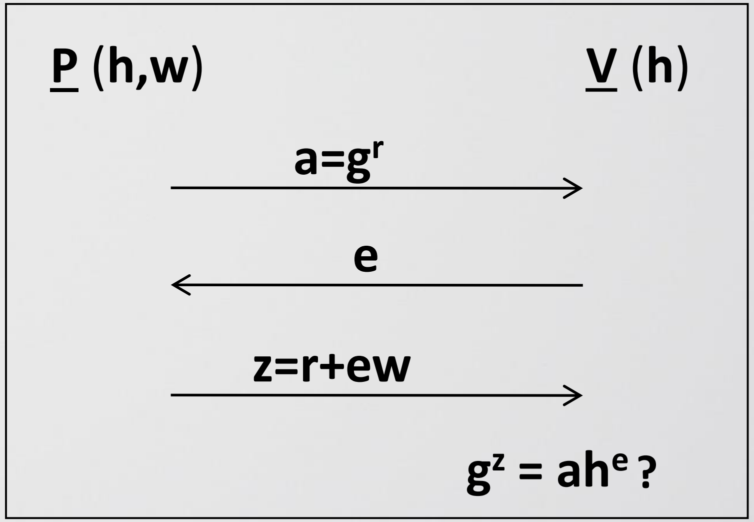
- An amazing concept; everything can be proven in zero knowledge
- Central to fundamental feasibility results of cryptography (e.g., the GMW compiler)
- But, can it be efficient?
  - It seems that zero-knowledge protocols for “interesting languages” are complicated and expensive
  - → Zero knowledge is often avoided

Sigma Protocols

- A way to obtain efficient zero knowledge
  - Many general tools
  - Many interesting languages, especially for arithmetic relations, can be proven with a sigma protocol

An Example – Schnorr's Protocol for Discrete Log

- Let G be a group of order q, with generator g
- P and V have input h$\in$G. P has w such that $g^w = h$
- P proves that to V that it knows $\mathrm{DLOG}_g(h)$
  - **P** chooses a random **r** and sends **$a=g^r$** to **V**
  - **V** sends **P** a random **e**$\in${**0,1**}$^{\mathbf{t}}$
  - **P** sends **z=r+ew** mod **q** to **V**
  - **V** checks that **$g^z = ah^e$**



P (h,w)

V (h)

$a=g^r$

e

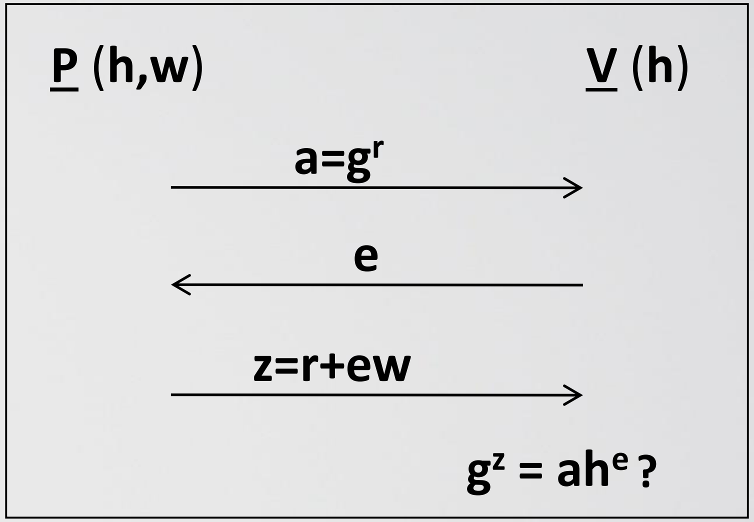
z=r+ew

$g^z = ah^e$?

Schnorr’s Protocol - Completeness

- Correctness:

$$g^z = g^{r+ew} = g^r(g^w)^e = ah^e$$



P (h,w) **V** (h)

$a=g^r$

e

z=r+ew

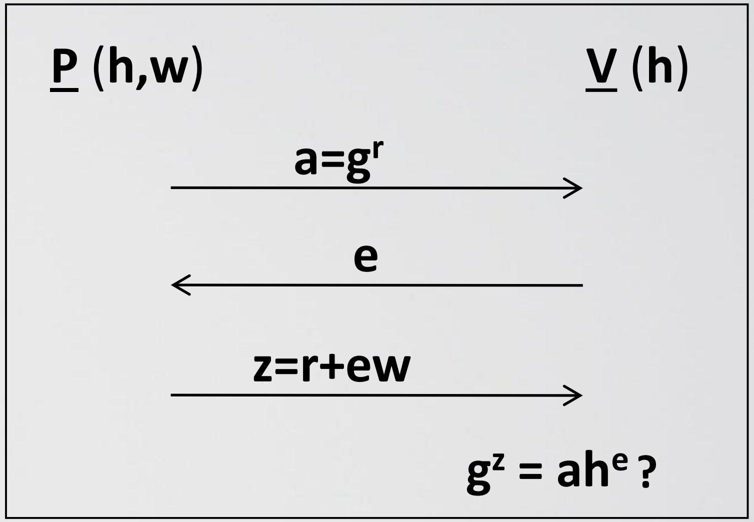
$g^z = ah^e$?

ZK Proof of Knowledge

- Prover P, verifier V, relation R
- P proves that it knows a witness w for which $(x,w) \in R$ without revealing anything
  - There exists an extractor **K** that obtains **w** such that **(x,w)**$\in$**R** from any **P*** with the same probability that **P*** convinces **V**

Schnorr’s Protocol – Proof of Knowledge

- Proof of knowledge
  - Assume **P** can answer **two** queries **e** and **e′** for the same **a**
  - Then, it holds that $g^z = ah^e$, $g^{z'}=ah^{e'}$
  - Dividing the two equations gives $g^{z-z'}=h^{e-e'}$
  - Therefore $h = g^{(z-z')/(e-e')}$
  - That is: $\mathrm{DLOG}_g(h) = (z-z')/(e-e')$
- Conclusion:
  - If **P** can answer with probability greater than $1/2^t$, then it must know the discrete log



P (h,w)

V (h)

$a=g^r$

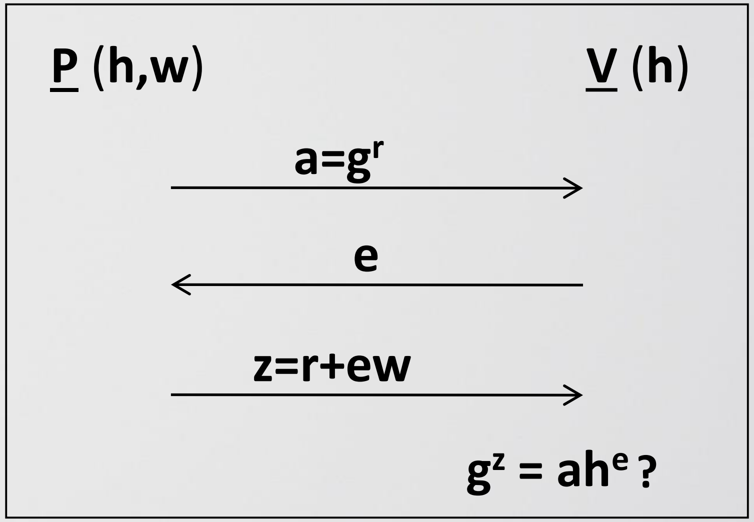
e

$z=r+ew$

$g^z = ah^e$?

Schnorr’s Protocol – Zero Knowledge

- What about zero knowledge? This does not seem easy.
  - ZK holds here if the verifier sends a **<u>random</u>** challenge **e**
  - This property is called “Honest-verifier zero knowledge”



<u>P</u> (h,w)

<u>V</u> (h)

$a=g^r$

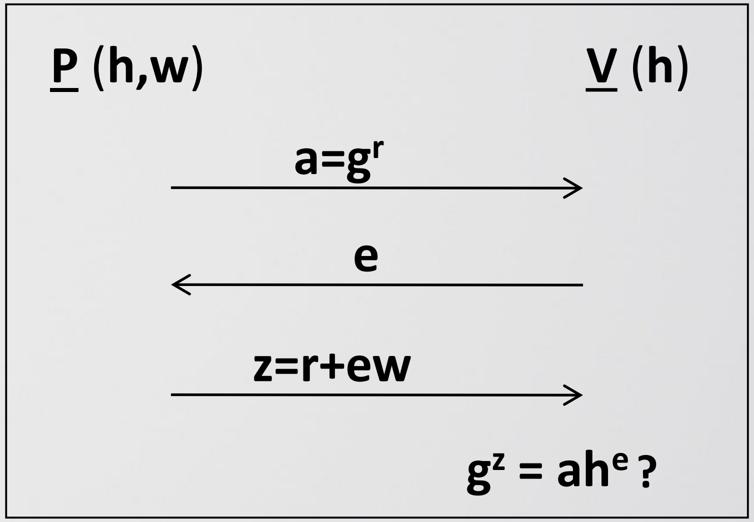
e

$z=r+ew$

$g^z = ah^e$?

Schnorr’s Protocol – Zero Knowledge

- What about zero knowledge? This does not seem easy.
  - ZK holds here if the verifier sends a **random** challenge **e**
  - This property is called “Honest-verifier zero knowledge”
- The simulation:
  - Choose a random **z** and **e**, and compute **$a = g^z h^{-e}$**
  - Clearly, (**a,e,z**) have the same distribution as in a real run. Namely, random values satisfying **$g^z = a \cdot h^e$**



P (h,w) **V** (h)

$a=g^r$ →

e ←

z=r+ew →

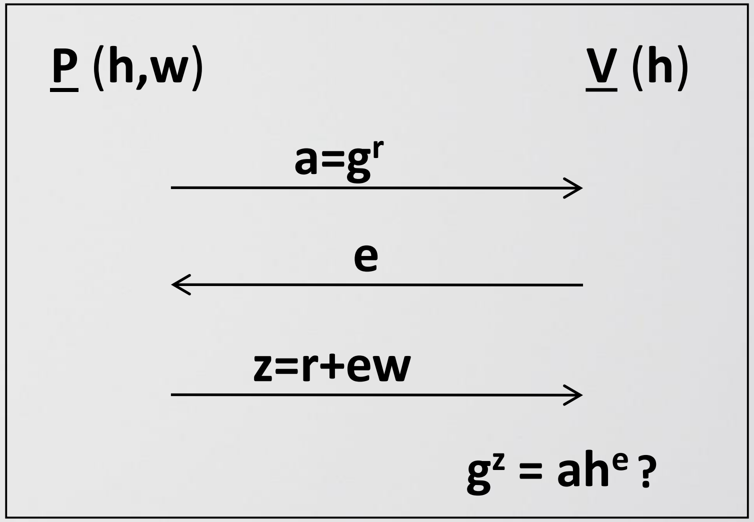
$g^z = ah^e$?

Schnorr's Protocol – Zero Knowledge

- What about zero knowledge? This does not seem easy.
  - ZK holds here if the verifier sends a **random** challenge **e**
  - This property is called "Honest-verifier zero knowledge"

- This is **not** a very strong guarantee, but we will see that it yields efficient general ZK.

- (Why does this only work for a verifier that chooses **e** at random?)



P (h,w) **V** (h)

$a=g^r$

e

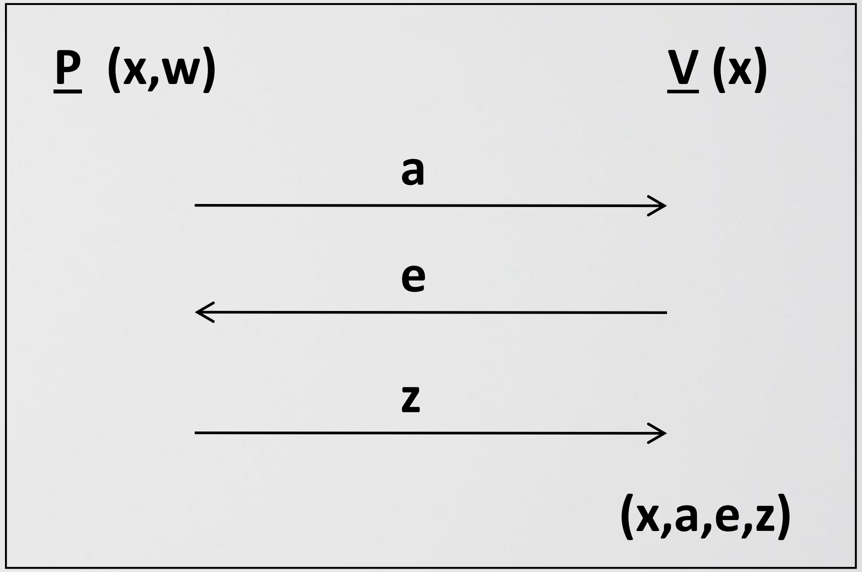
$z=r+ew$

$g^z = ah^e$?

Definitions

- Sigma protocol template
  - **Common input: P** and **V** both have **x**
  - **Private input: P** has **w** such that (**x**,**w**)∈**R**

  - **Three-round protocol:**
    - **P** sends a message **a**
    - **V** sends a random **t**-bit string **e**
    - **P** sends a reply **z**
    - **V** accepts based solely on (**x**,**a**,**e**,**z**)



P (x,w)

V (x)

a

e

z

(x,a,e,z)

Definitions

- Completeness: as usual in ZK
- Special soundness:
  - There exists an efficient extractor **A** that given any **x** and pair of transcripts $(\mathbf{a},\mathbf{e},\mathbf{z}),(\mathbf{a},\mathbf{e}',\mathbf{z}')$ with $\mathbf{e}\neq\mathbf{e}'$ outputs **w** s.t. $(\mathbf{x},\mathbf{w})\in\mathbf{R}$
- Special honest-verifier ZK
  - There exists an efficient simulator **S** that given any **x** and **e** outputs an accepting transcript $(\mathbf{a},\mathbf{e},\mathbf{z})$ which is distributed exactly like a real execution where **V** sends **e**

Another example: Sigma Protocol for a DH Tuple

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

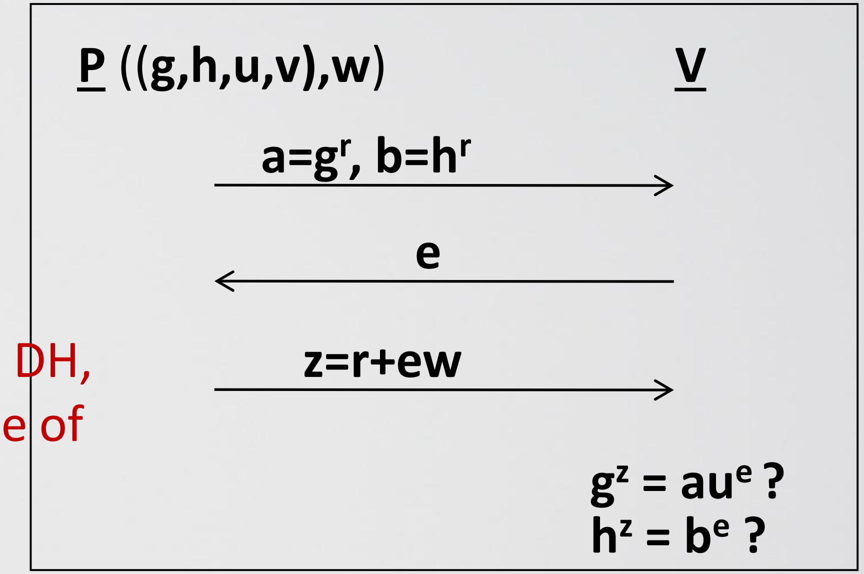
- Protocol
  - **P** chooses a random **r** and sends $\mathbf{a}=\mathbf{g}^{\mathbf{r}}$, $\mathbf{b}=\mathbf{h}^{\mathbf{r}}$
  - **V** sends a random **e**
  - **P** sends $\mathbf{z}=\mathbf{r}+\mathbf{e}\mathbf{w}$ mod **q**
  - **V** checks that $\mathbf{g}^{\mathbf{z}}=\mathbf{a}\mathbf{u}^{\mathbf{e}}$, $\mathbf{h}^{\mathbf{z}}=\mathbf{b}\mathbf{v}^{\mathbf{e}}$

Sigma Protocol for Proving a DH Tuple

- Completeness: as in DLOG
- Special soundness:
  - (Like DLOG) 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

    $\log_g u = \log_h v = w = (z-z')/(e-e')$

    In addition to proving DH, also proves knowledge of the discrete log
- 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**

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

$\xleftarrow{e}$

$\xrightarrow{z=r+ew}$

$g^z = au^e$?
$h^z = b^e$?

Basic Properties of Sigma Protocols

- Any sigma protocol is an interactive proof with soundness error 2^{-t}
- Properties of sigma protocols are invariant under parallel composition
- Any sigma protocol is a proof of knowledge [BG92] with error 2^{-t}
  - The difference between the probability that $\mathbf{P}^*$ convinces **V** and the probability that an extractor **K** obtains a witness is at most $\mathbf{2^{-t}}$
  - Proof needs some work

Sigma Protocols

- Very efficient honest–verifier ZK three-round protocols
- Can be applied to many problems
  - Almost all Dlog/DH statements (?)
  - Proving that a commitment is for a specific value
  - Proving that a Paillier encryption is of zero
  - and many other applications…

Non-Interactivity using the Fiat-Shamir Paradigm

- To prove a statement **x** non-interactively
  - Generate **a**
  - (Instead of receiving **e**) compute **e=H(a,x)**
  - Compute **z**
  - Send (**a,e,z**)
- The challenge **e** must be long (128 bits or more)
- No need to worry anymore about honesty of the verifier
- But, only secure in the random oracle model

Tools for Sigma Protocols

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

Proving Compound Statements

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 knowledge of many discrete logs can be done in much less time than running all proofs independently
    - Batch all into one tuple and prove (how?)

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 of the two proofs

OR of Sigma Protocols

- The template for proving x_0 or x_1:
  - **P** sends two first messages (**a_0,a_1**)
  - **V** sends a single challenge **e**
  - **P** replies with
    - Two challenges **e_0,e_1** s.t. **$e_0 \oplus e_1 = e$**
    - Two final messages **z_0,z_1**
  - **V** accepts if **$e_0 \oplus e_1 = e$** and (**a_0,e_0,z_0**),(**a_1,e_1,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,$\mathbf{e_1}$ s.t. $e_0 = e \oplus e_1$ and with z_0,z_1
  - **P** already has $\mathbf{z_1}$ and can compute $\mathbf{z_0}$ using the witness
- Special soundness
  - If P doesn't know a witness for $\mathbf{x_1}$, it can only answer for a single $\mathbf{e_1}$
  - This means that for $\mathbf{x_0}$, the challenge **e** defines a random challenge $\mathbf{e_0}$, like in a regular proof

OR of Sigma Protocols

- Special soundness
  - Relative to first message ($\mathbf{a_0,a_1}$), and two different verifier challenges $\mathbf{e,e'}$, it holds that either $\mathbf{e_0 \neq e'_0}$ or $\mathbf{e_1 \neq e'_1}$
  - Thus, for at least one of the statements we can use the special soundness of the single protocol to compute a witness for that statement, which is also a witness for the OR statement.
- Honest verifier ZK
  - The simulation can choose both $\mathbf{e_0,e_1}$, so no problem.
- Note that it is possible to prove an OR of different statements using different protocols

OR of Many Statements

- Prove k out of n statements $x_1,\ldots,x_n$

Main tool: k-out-of-n secret sharing

- Let F be a field.
- Basic facts from algerbra:
  - Any d+1 pairs (a_i , b_i) define a unique polynomial P of degree d, s.t. $P(a_i)=b_i$. (assuming $d < |F|$)
  - This polynomial can be found using interpolation
  - Given a polynomial that was interpolated from random points, it is impossible to identify the points which were used to interpolate it.

OR of Many Statements

- Sigma protocol for k out of n statements $x_1,\ldots,x_n$
  - **A** = set of indices that prover knows how to prove $|A|=k$
  - **B** = all other indices $|B|=n-k$
  - Will use a polynomial with n-k+1 degrees of freedom
  - Field elements 1,2,…,n. Polynomial **f** of degree n-k
- First step:
  - For every $\mathbf{i \in B}$, prover generates $(\mathbf{a_i,e_i,z_i})$ using SIM
  - For every $\mathbf{j \in A}$, prover generates $\mathbf{a_j}$ as in protocol
  - Prover sends $(\mathbf{a_1,\ldots,a_n})$

OR of Many Statements

- Prover sent $\mathbf{(a_1,\ldots,a_n)}$
- Verifier sends a random field element $\mathbf{e \in F}$
- Prover finds the (only) polynomial f of degree n-k passing through all (i,e_i) and $(0,e)$ (for $i \in B$)
  - For every $\mathbf{j \in A}$, the prover computes $\mathbf{e_j = f(j)}$, and computes $\mathbf{z_j}$ as in the protocol, based on transcript $\mathbf{a_j, e_j}$
  - For every $\mathbf{j \in B}$, the prover uses $\mathbf{e_i}$ (for which it already prepared an answer using SIM)
- The verifier verifies that all $\mathbf{e_i}$ values are on a polynomial of degree n-k

OR of Many Statements

- Special soundness:
  - Suppose that the prover can prove less than k statements
  - So for more than n-k statements it can only answer a single query (per query)
  - These queries define a polynomial of degree n-k
  - These queries will be asked only if the verifier chooses to use e=f(0), which happens with probability 1/|F|

General Compound Statements

- These techniques 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

ZK from Sigma Protocols

- In ZK proofs the verifier is not necessarily honest
  - The problem is that it might choose its challenge based on the first message of the verifier
- The verifier might set its challenge based on the first message it received from the prover
- The simulation for honest verifiers will no longer work

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.)

ZK from Sigma Protocols

- The basic idea
  - Have **V** first commit to its challenge **e** using a perfectly-hiding commitment
- The protocol
  1. **P** sends the first message α of the commit protocol
  2. **V** sends a commitment c=$\mathbf{Com}_{\alpha}$**(e;r)**
  3. **P** sends a message **a**
  4. **V** opens the commitment by sending (**e,r**)
  5. **P** checks that c=$\mathbf{Com}_{\alpha}$**(e;r)** and if so sends a reply **z**
  6. **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 the transcript of the protocol:
    - **V** commits.
    - Send to **V** a message **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**

What happens if V refuses to decommit?

- V might refuse, with probability **p**, to decommit to **e**.
- Since the simulation chooses a random **a**, we can get V to open the commitment after **1/p** attempts (in expectation)

Implementing Commitments: Pedersen

- Highly efficient perfectly-hiding commitments (two exponentiations for string commit)
  - **Parameters:** generator $\mathbf{g}$, order $\mathbf{q}$
  - **Commit protocol** (commit to $\mathbf{x}$):
    - Receiver chooses random $\mathbf{k}$ and sends $\mathbf{h=g^k}$
    - Sender sends $\mathbf{c=g^r h^x}$, for random $\mathbf{r}$
  - **Perfectly hiding:**
    - For every $\mathbf{y}$ there exists $\mathbf{s}$ s.t. $\mathbf{g^s h^y} = c = \mathbf{g^r h^x}$
  - **Computationally binding:**
    - If sender can open commitment in two ways, i.e. find $(\mathbf{x,r}),(\mathbf{y,s})$ s.t. $\mathbf{g^r h^x=g^s h^y}$, then it can also compute the discrete log $\mathbf{k} = (\mathbf{r-s})/(\mathbf{y-x})$ $\mathbf{mod\ q}$

Efficiency of ZK

- Using Pedersen commitments, the entire DLOG proof costs only 5 additional group exponentiations

ZKPoK from Sigma Protocols

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 there might be a prover which chooses its message **a** based on the commitment to **e**, and so when the extractor changes the commitment the prover changes **a**

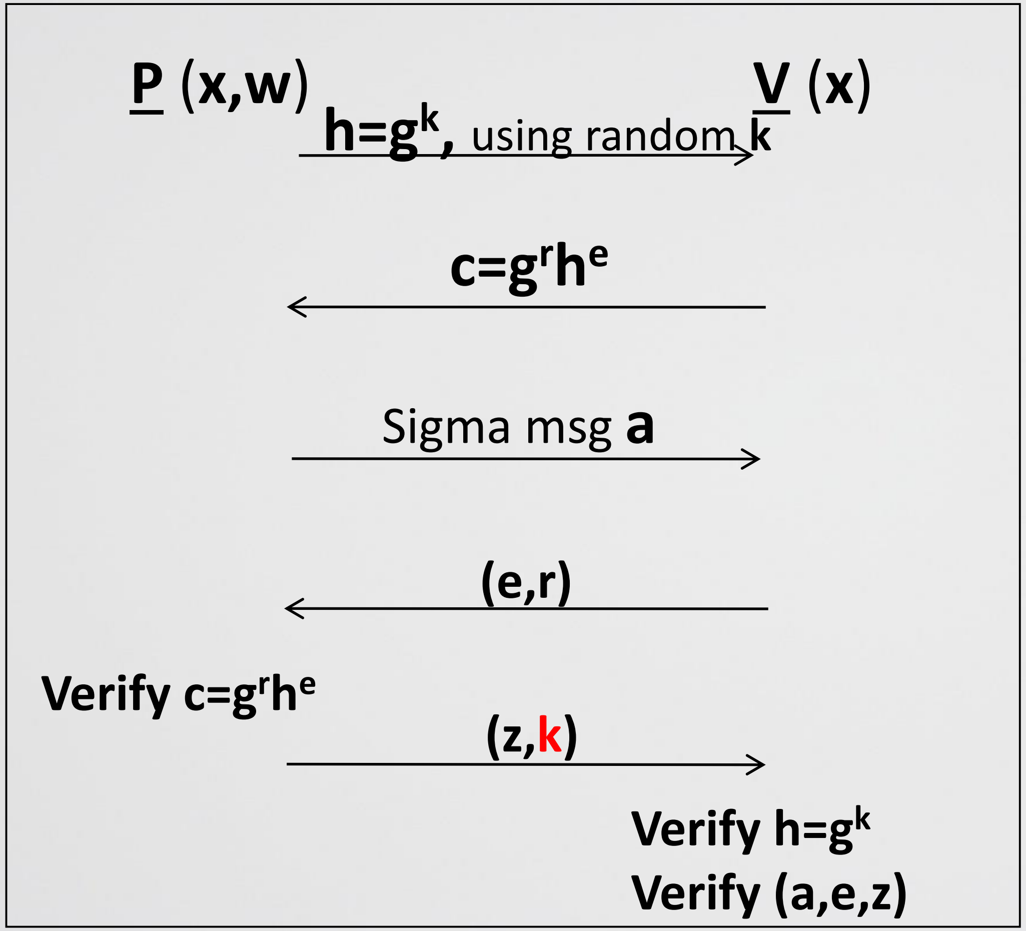
ZKPOK from Sigma Protocols

- Solution: use a trapdoor (equivocal) commitment
  - Namely, 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 **r+kx = s+ky mod q**
  - This is easy if **k** is known: compute **s = r+k(x-y) mod q**

ZKPOK from Sigma Protocols

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

ZKPOK from Sigma Protocols



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

$h=g^k$, using random k

$c=g^r h^e$

Sigma msg **a**

(e,r)

Verify $c=g^r h^e$

(z,k)

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)

Side note: Constructing Commitments from Sigma Protocols

- Based on a hard relation R
  - A generator G outputs **(x,w)∈R**
  - But for every PPT algorithm A it is hard to find **w** given **x**, namely Pr[A(x)∈R] is negligible

- Example
  - The generator computes $\mathbf{h=g^r}$ for a random **r**

The Commitment Scheme

- Commitment to a string $\mathbf{e} \in \{0,1\}^t$
  - The **receiver** samples a hard (**x,w**), and sends **x**
  - **Committer** runs the sigma protocol simulator on (**x,e**), gets (**a,e,z**) and sends **a** as the commitment
- Decommitment:
  - Committer sends (**a,e,z**)
  - Decommitter verifies that is accepting proof for **x**
- Hiding: By HVZK, the commitment **a** is independent of **e**
- Binding: Decommitting to two **e,e′** for the same **a** means finding **w**

This is a Trapdoor Commitment

- The scheme is actually a trapdoor commitment scheme
  - **w** is a trapdoor
  - Given **w**, can decommit to any value by running the **real** prover and not the simulator

Summary

- Don’t be afraid of using zero knowledge
  - Using sigma protocols, we can get very efficient ZK
- Sigma protocols are very useful:
  - Efficient ZK
  - Efficient ZKPOK
  - Efficient NIZK in the random oracle model
  - Commitments and trapdoor commitments
  - More…