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 $\mathbf{x \notin L}$, for any $\mathbf{P^*}$
    - Computational soundness: only holds when $\mathbf{P^*}$ 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 $\mathbf{f}_{\mathbf{zk}}((\mathbf{x},\mathbf{w}),\mathbf{x}) = (\text{-},\mathbf{R}(\mathbf{x},\mathbf{w}))$

Zero Knowledge

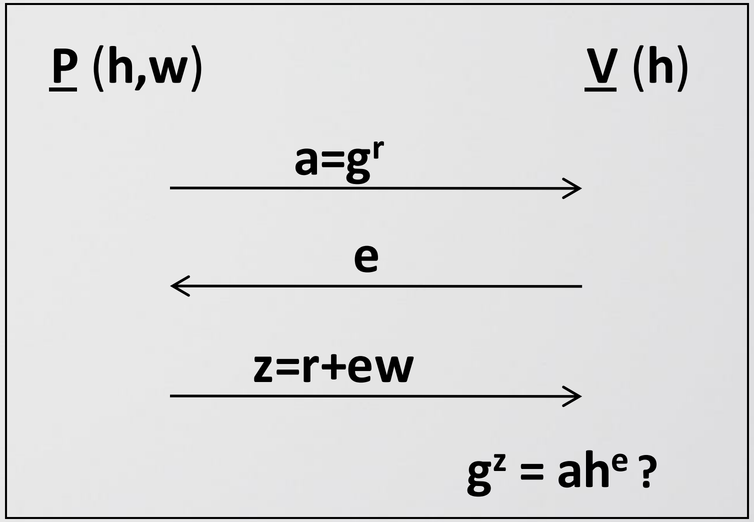
- An amazing concept; everything can be proven in zero knowledge
- Central to fundamental feasibility results of cryptography (e.g., the GMW compiler)
- But, can it be efficient?
  - It seems that zero-knowledge protocols for "interesting languages" are complicated and expensive
  - → Zero knowledge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

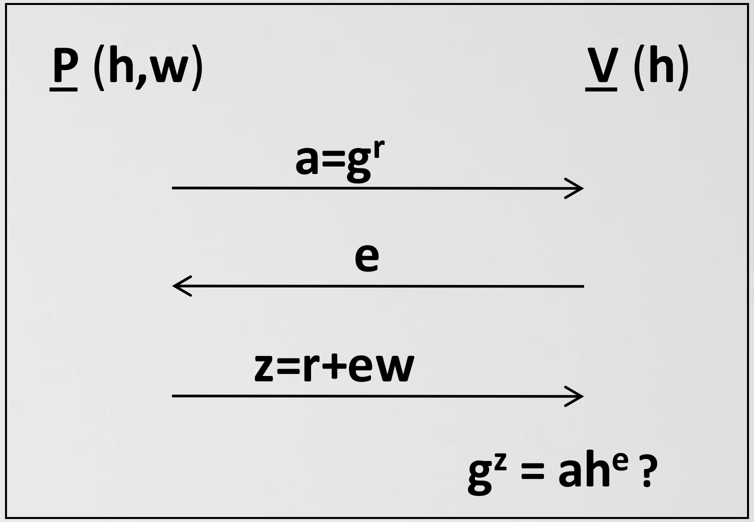
- Let G be a group of order q, with generator g
- P and V have input $h \in G$. P has w such that $g^w = h$
- P proves that to V that it knows $DLOG_g(h)$
  - **P** chooses a random **r** and sends $a=g^r$ to **V**
  - **V** sends **P** a random $e \in \{0,1\}^t$
  - **P** sends **z=r+ew** mod **q**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



Schnorr's Protocol - Completeness

- Correctness:

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

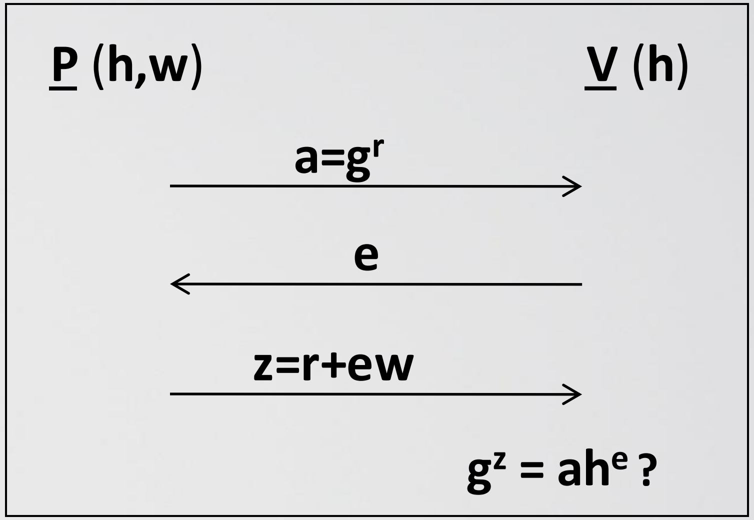


ZK Proof of Knowledge

- 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**

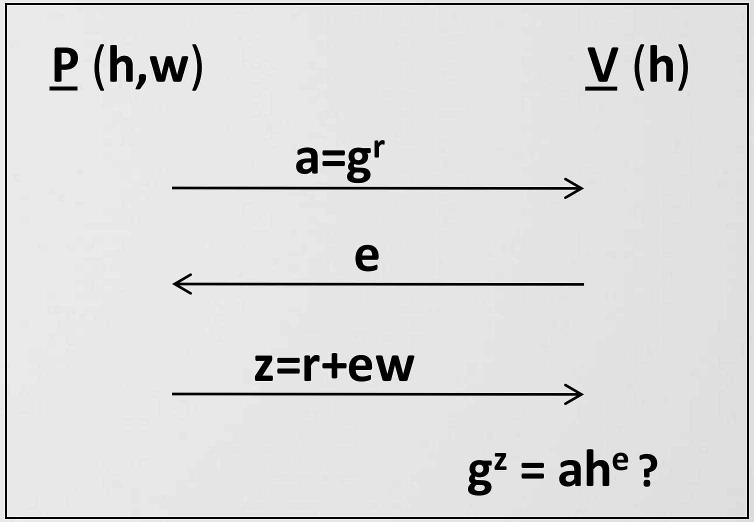
Schnorr's Protocol – Proof of Knowledge

- 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: $DLOG_g(h) = (z-z')/(e-e')$
- Conclusion:
  - If **P** can answer with probability greater than $1/2^t$, then it must know the discrete log



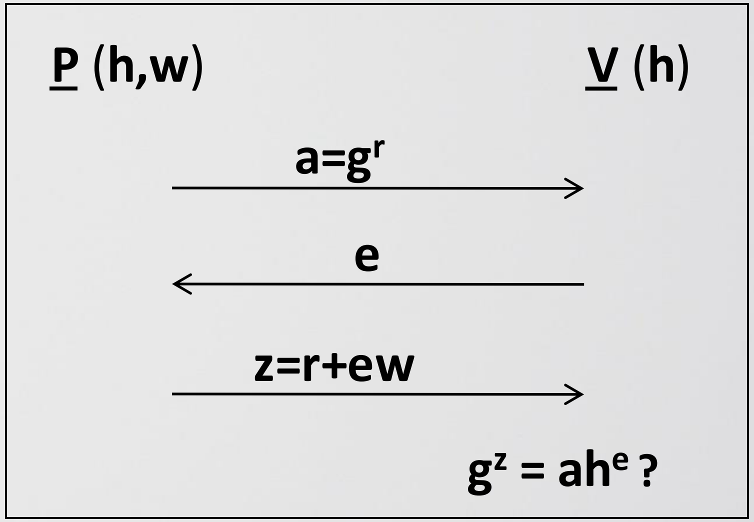
Schnorr's Protocol – Zero Knowledge

- 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"



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 $\mathbf{a = g^z h^{-e}}$
  - Clearly, (**a,e,z**) have the same distribution as in a real run. Namely, random values satisfying $\mathbf{g^z = a \cdot h^e}$

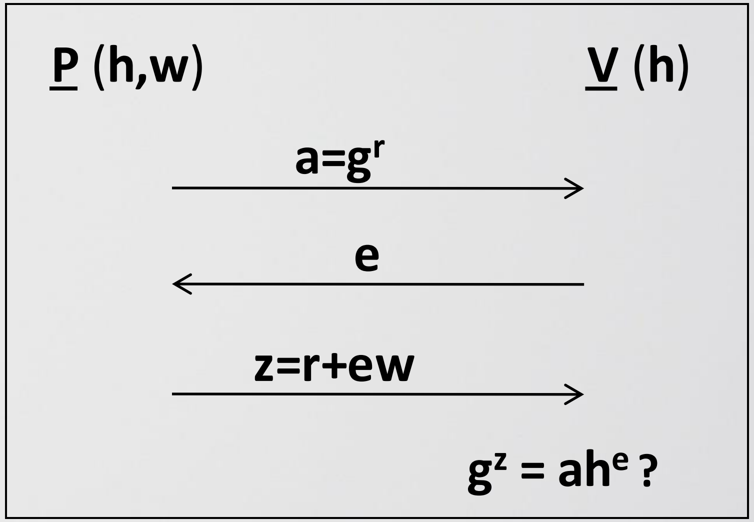


Schnorr's Protocol – Zero Knowledge

- 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.

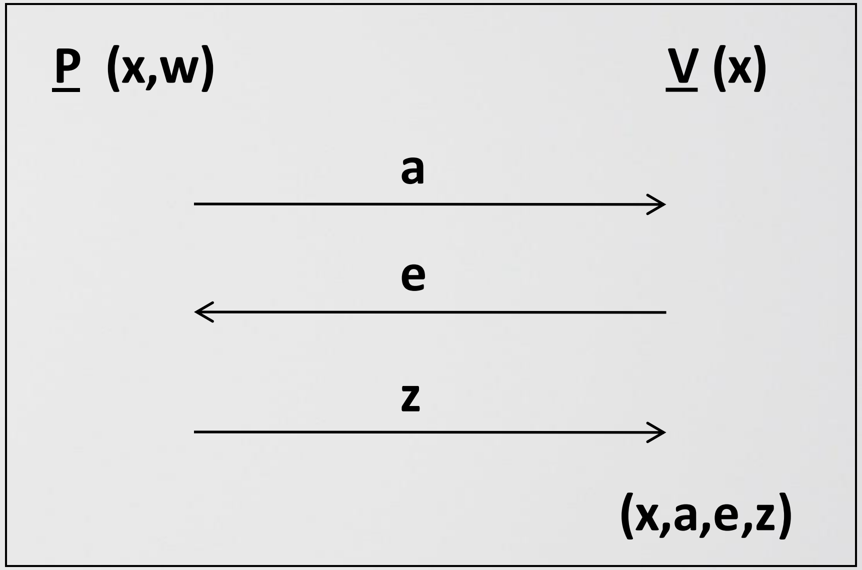
- (Why does this only work for a verifier that chooses **e** at random?)



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 <u>random</u> **t**-bit string **e**
    - **P** sends a reply **z**
    - **V** accepts based solely on (**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 (**a,e,z**),(**a,e′,z′**) with **e≠e′** outputs **w** s.t. (**x,w**)∈**R**
- **Special honest-verifier ZK**
  - There exists an efficient simulator **S** that given any **x** and **e** outputs an accepting transcript (**a,e,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
  - **(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, of equality of dlogs, 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 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**

$a=g^r, b=h^r$ $\longrightarrow$

$\longleftarrow$ e

$z=r+ew$ $\longrightarrow$

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

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 $(\mathbf{a_0},\mathbf{a_1})$
  - **V** sends a single challenge $\mathbf{e}$

  - **P** replies with
    - Two challenges $\mathbf{e_0},\mathbf{e_1}$ s.t. $\mathbf{e_0 \oplus e_1 = e}$
    - Two final messages $\mathbf{z_0},\mathbf{z_1}$

  - **V** accepts if $\mathbf{e_0 \oplus e_1 = e}$ 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 ($\mathbf{a_0,a_1}$)
  - 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 ($\mathbf{a_1,e_1,z_1}$)
  - **P** sends ($\mathbf{a_0,a_1}$)
- **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
- <u>Special soundness</u>
  - 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.

Interlude

Sudoku

9						2		7
	1			2	8			
7	4				1			
						6		9
3			6	8	7			4
2		4						
			3				9	8
			8	1			5	
1		9						6

297-5240 - www.Sudoweb.com - Free Sudoku and ebook

"Cryptographic and Physical Zero-Knowledge Proof Systems for Solutions of Sudoku Puzzles"

Ronen Gradwohl, Moni Naor, Benny Pinkas, Guy N. Rothblum

(we will talk about an nxn puzzle)

Alice and Bob solve a Sudoku puzzle

- Bob: I solved it!
- Alice: I don’t believe you

A cryptographic protocol for Sudoku based on coloring

- An adaptation of a known protocol for 3-colorability
- The protocol:
  - P chooses a random permutation $\sigma:\{1,\ldots,n\} \to \{1,\ldots,n\}$
  - $\forall$ entry with value v, P sends to V a commitment to $\sigma(v)$
  - V chooses at random one of 3n+1 options:
    - a specific row, column or subgrid
    - or “filled-in entries”
  - P opens the commitments corresponding to V’s choice.

- Completeness is trivial

Example puzzle

		8		1		7	6	
	3							1
			6		4			2
3			1					
	4	5	7		2	1	9	
					6			5
2			9		5			
4							2	
	9	1		6		4		

Daily SuDoku: Tue 1-May-2007 very hard

x	σ(x)
1	4
2	8
3	3
4	5
5	1
6	9
7	6
8	7
9	2

Soundness

- If P can answer all 3n+1 challenges then there must be a solution to the puzzle
- Therefore if there is no solution to the puzzle, there must be at least one challenge which P cannot answer.
- V therefore rejects with probability $\geq 1/(3n+1)$

This soundness error (1-1/(3n+1)) seems a bit high… (there is a better cryptographic protocol with a constant soundness error)

Zero-knowledge

- Zero-knowledge follows a standard argument:
  - The distribution of P’s answer is efficiently computable given the puzzle and the challenge.
  - The number of possible challenges is only 3n+1.
  - Simulator guesses challenge and prepares commitments which answer challenge. If V asks this challenge, the simulator can answer. Otherwise, try again.

Knowledge Extraction

- Given P’s commitments ask it to open one challenge, then rewind the protocol, ask it to open to another challenge, and so on.
  - This reveals the solution.

Physical protocols ?

- Commitments can be implemented physically (using envelopes)
  - Therefore the cryptographic protocol can be physically implemented
  - But this is not easy for humans (compute a permutation σ? repeat $O(n\log(1/\varepsilon))$ times?)

First protocol using cards ("one card per cell")

- The protocol
  1. Prepare an $n \times n$ board.
  2. P assigns a card with the right value to every cell.
  3. P puts the card *face up* if the cell value is "filled-in", and *face down* if the cell is part of the solution.
  4. V chooses one of "Rows" / "Columns" / "Sub-grids".
  5. P arranges the cards in n sets according to V's choice.
  6. P shuffles the cards in each of the n sets, and shows that each set contains the values 1…n.

Analysis

- **Completeness:** perfect.
- **Soundness:** If P does not know a solution, then it cannot answer at least one of V's 3 choices.
  - Soundness error: 2/3 *(might be too high)*
- **Zero-knowledge:** the simulator
  - Puts arbitrary cards face down
  - Follows V's instructions, but before opening the packets it replaces the cards with cards with the right values.
- **Proof of knowledge:**
  - P puts the cards face down. The knowledge extractor opens them.

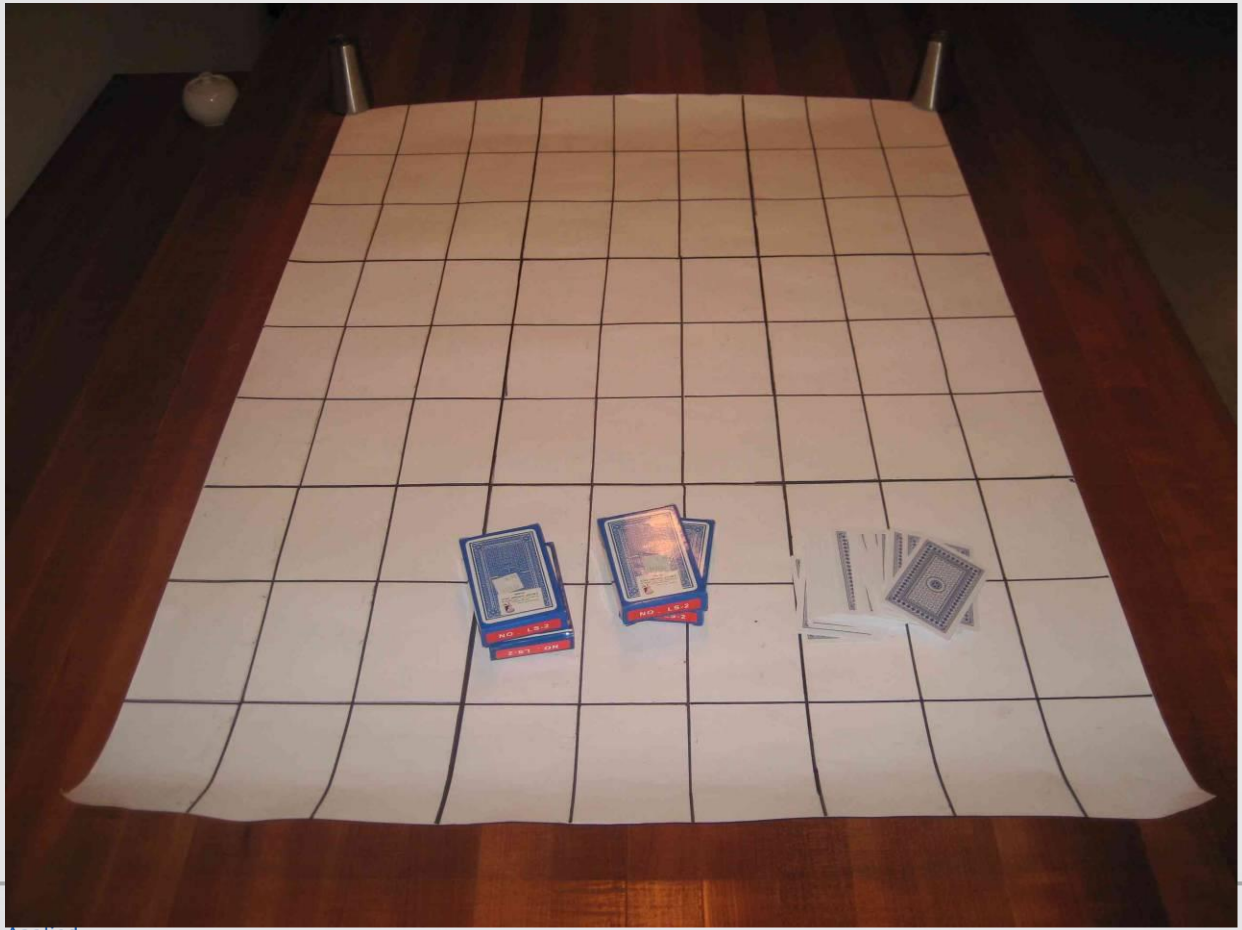
Comparing different physical protocols

- Possible criteria:
  - Number of cards
  - Number of shuffles
  - Soundness error

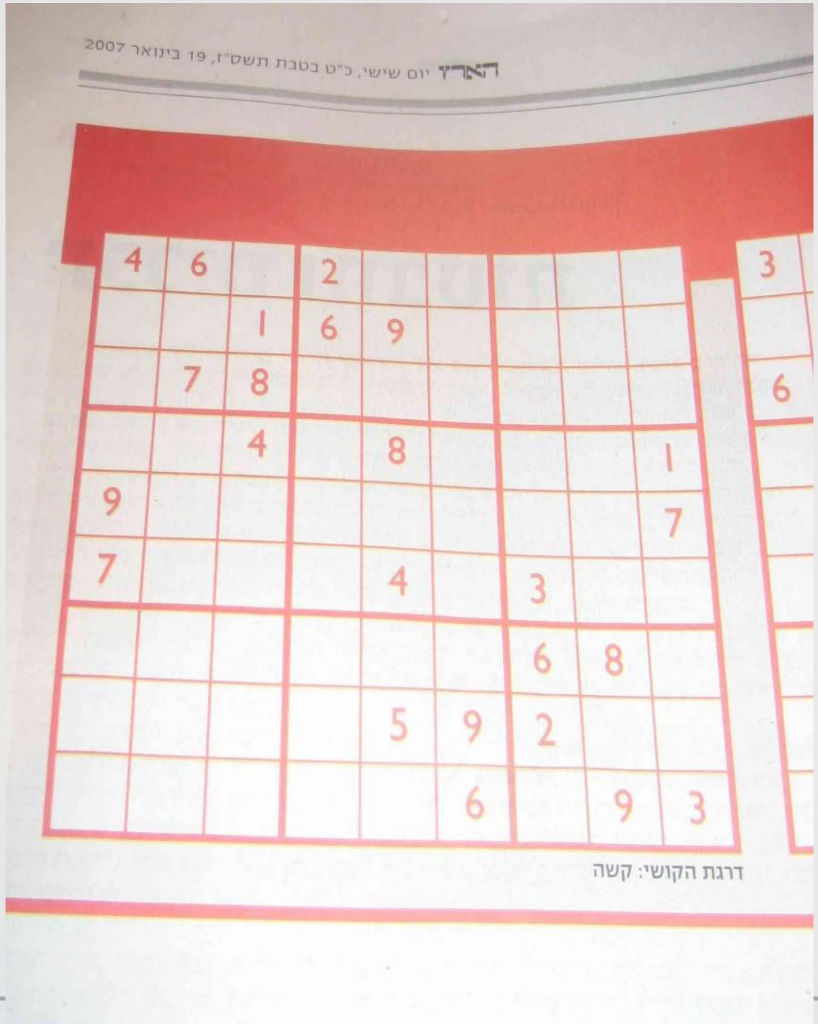
	# of cards	shuffles	soundness error
Protocol 1: “one card per cell”	n^2	n	2/3
Protocol 2: “all packets”	$3n^2$	3n	1/9
Protocol 3: “aggregate packets”	$3n^2$	c-1	1/9+8/(9c)

Protocol 2: "all packets"

- Equipment: n×n grid, playing cards.

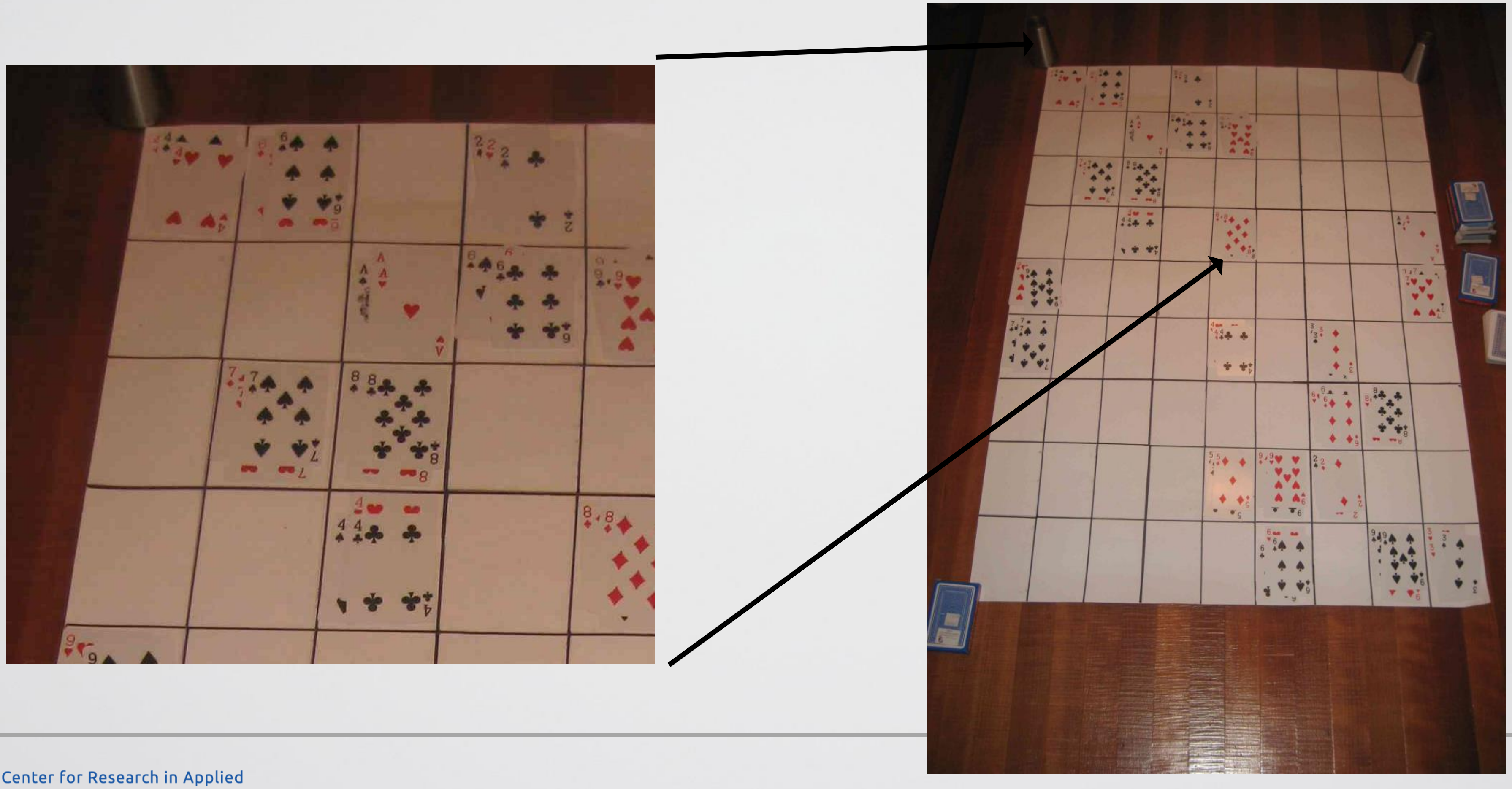


The puzzle



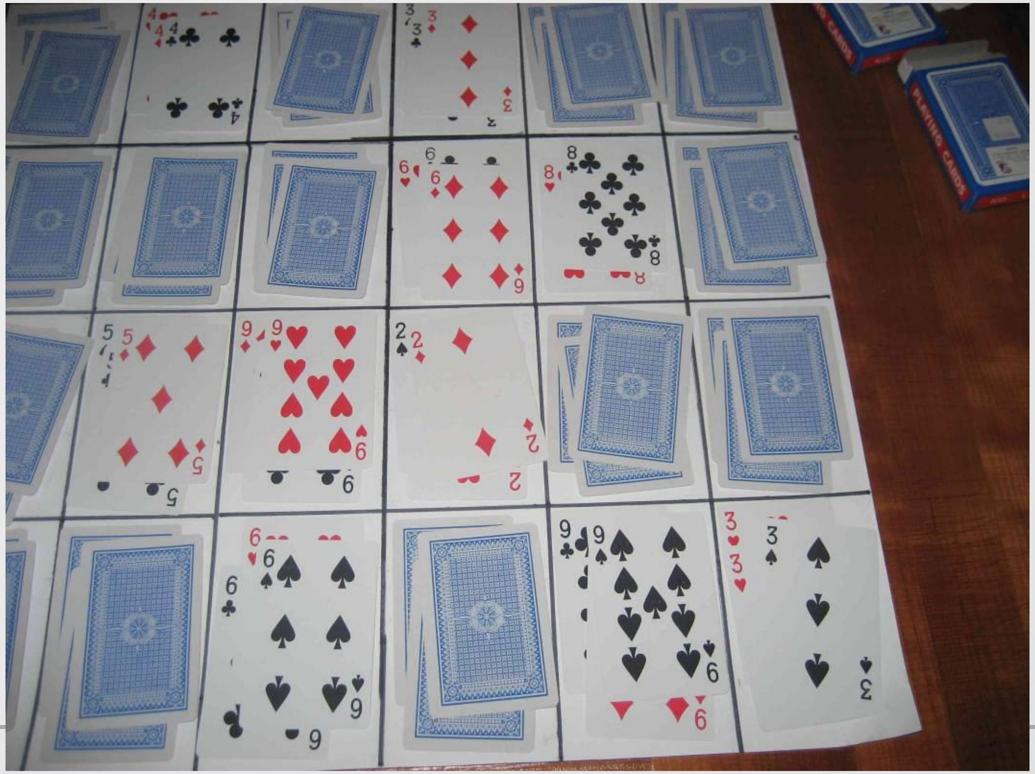
Step 1

P places **three** faced up cards on filled-in cells



Step 2

P places three identical cards (according to his solution), face down, on the remaining cells.



Step 3

- V generates packets for columns by picking a random card from every cell
- She then generates similar packets for rows

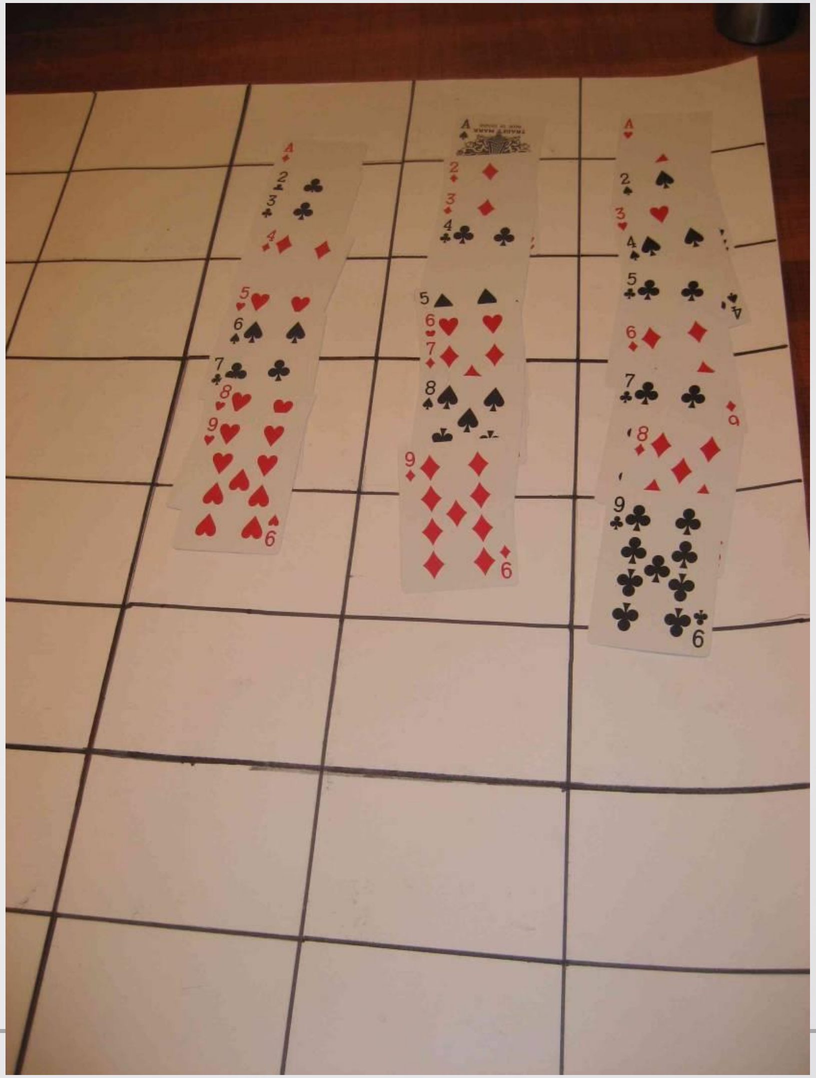
Step 4

- There are now packets for every row, column and sub-grid.

- P turns the cards upside down and shuffles them

Final step

- V opens the packets and verifies that each contains the numbers 1…9.
  - If not, V rejects.



Properties

- **Completeness:** trivial
- **Zero-knowledge:**
  - The simulator places arbitrary cards on the board. It follows the protocol but before handing the shuffled packets to V is replaces them with packets containing 1…9.
- **Knowledge extractor:**
  - Simply open the cards which were placed on the board.
- **Overhead:**
  - $3n^2$ cards (243 cards)
  - 3n shuffles (27 shuffles)

Soundness

- A simple proof that the soundness error $\leq 1/3$
  - Assume P does not know a solution.
  - If P places 3 identical cards on each cell, he is caught with probability 1.
  - There is therefore a cell *C* in which not all cards are equal: one card ("*y*") must be different than the two other cards.
  - Suppose that V assigned the cards of all other cells *but C* to rows/columns/sub-grids.
  - V will only accept if there is only a single packet that needs the card "*y*", and "y" is indeed assigned to it.
  - This happens with probability $\leq 1/3$.

Soundness

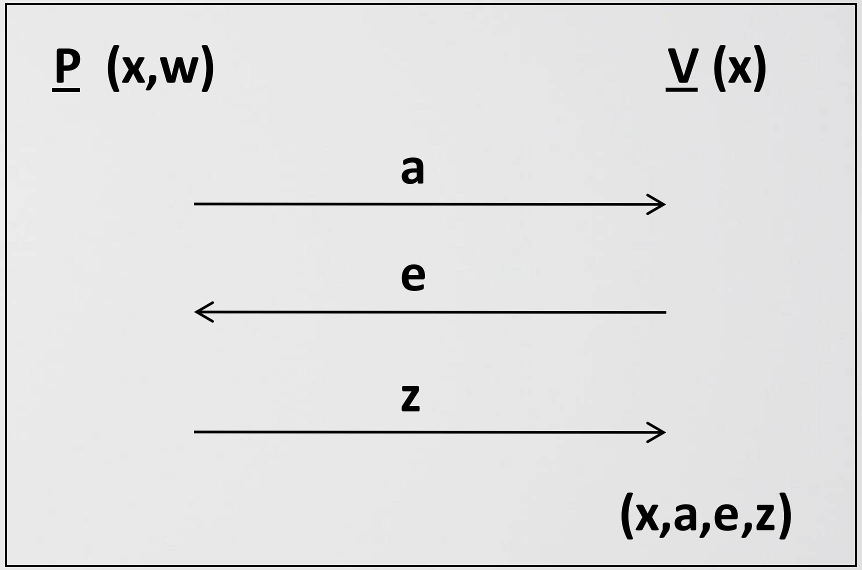
- It is possible to show that the **soundness error ≤ 1/9**:
  - The total number of cards from each value must be the same.
  - Therefore there must be at least **two** such cells in which P puts one card which is different than the others.
- The proof uses this fact
  - It assumes that all cards, except for these two cells, were assigned to packets.
  - It shows that the probability of generating balanced decks is ≤ 1/9.

ZK from Sigma Protocols

Sigma Protocols

- 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)**



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 prover

- The simulation for honest verifiers will no longer work

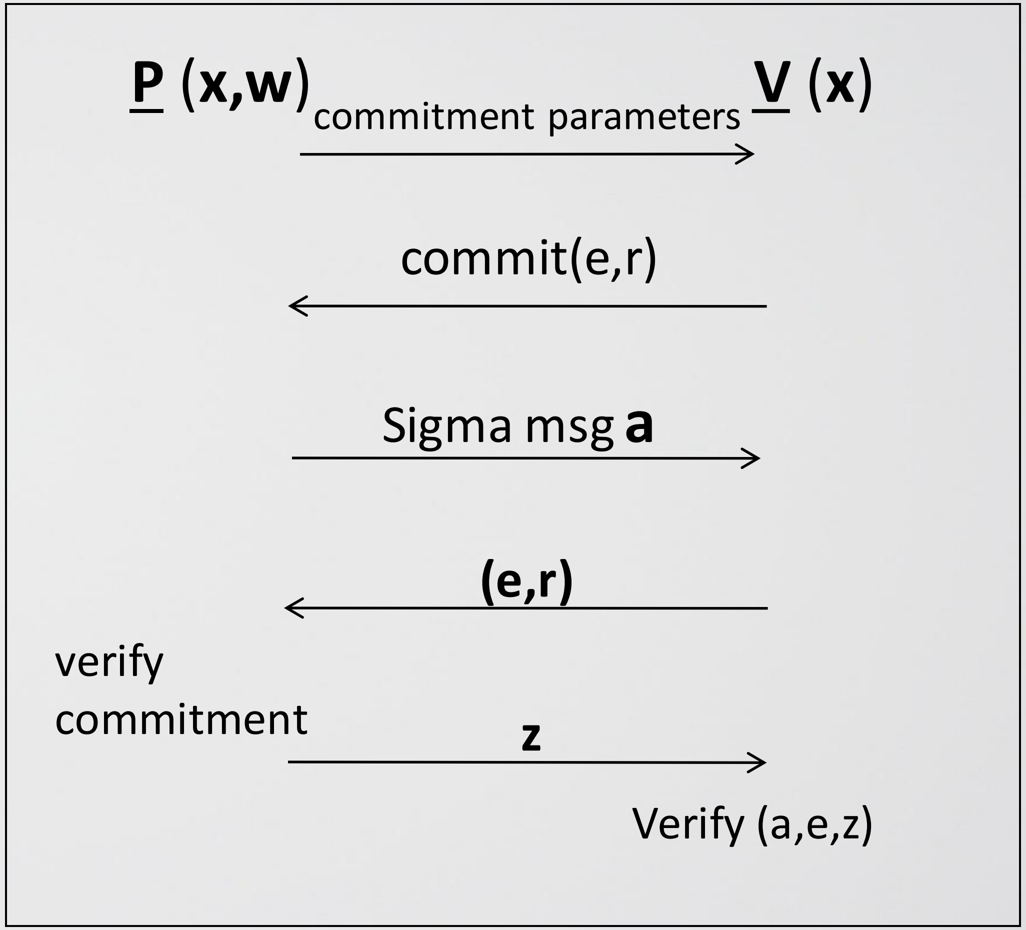
ZK from Sigma Protocols

- 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



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**
- 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

- Efficient perfectly-hiding commitments
  - **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**
  - **Perfectly hiding:**
    - For every **y** there exists **s** s.t. $g^s h^y = c = g^r h^x$
  - **Computationally 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 it can also compute the discrete log $k = (r-s)/(y-x) \bmod q$

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 a different first message **a** for every commitment string
    - 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**

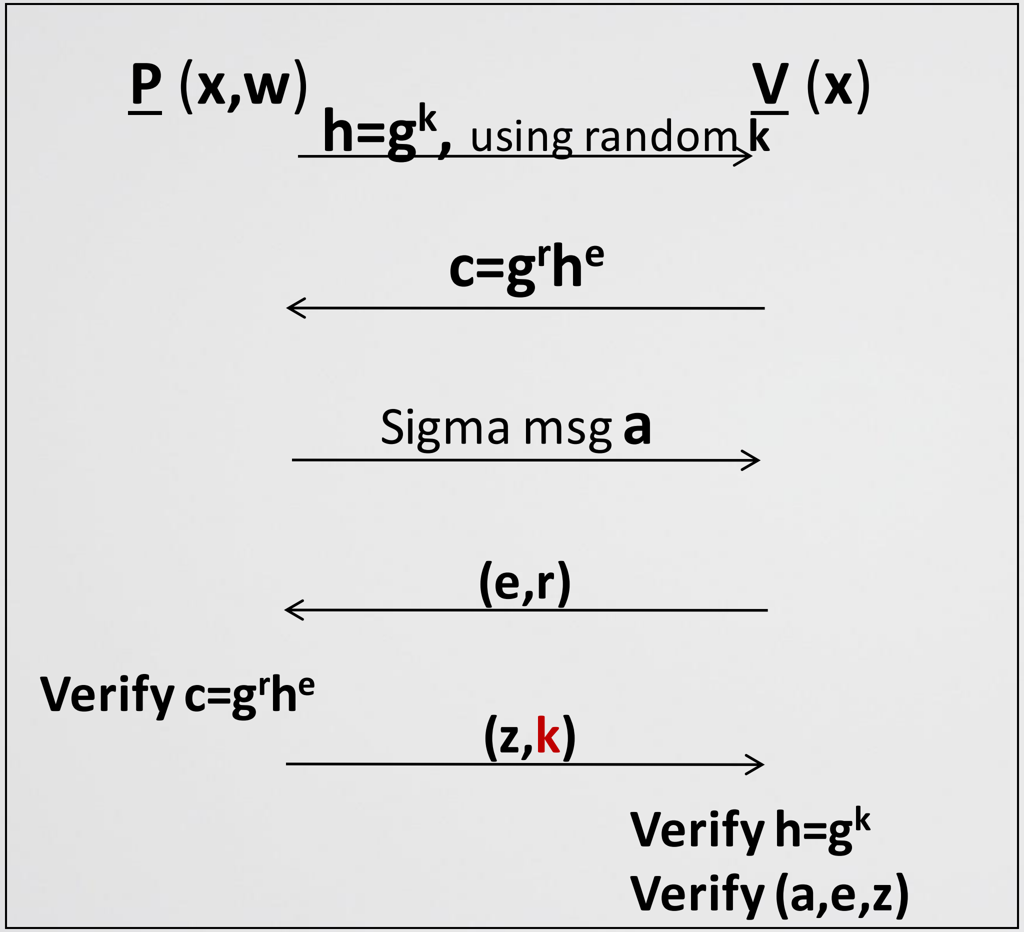
ZKPOK from Sigma Protocols

- 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 $\mathbf{r+kx = s+ky \bmod q}$
  - 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 (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 sends the **trapdoor** for the commitment
  6. **V** accepts if the **trapdoor** is correct and (**x,a,e,z**) is accepting

ZKPOK from Sigma Protocols



- The trapdoor k enables V to open the commitment to any value.
- But this does not help v since it receives k after it already opened the commitment.

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

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

Side note: Constructing Commitments from Sigma Protocols

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

- Example
  - The generator computes $\mathbf{h}=\mathbf{g}^{\mathbf{r}}$ 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…