FIT5124 Advanced Topics in Security

Lecture 5: Secure Computation Protocols I – Zero-Knowledge Proofs

Ron Steinfeld
Clayton School of IT
Monash University

April 2015

New topic: Secure Computation Protocols

Secure Computation Protocols: How to achieve more complex security requirements beyond basic confidentiality or integrity?

We will look at two topics:

- **Privacy in authentication and protocol integrity** (today's lecture): Zero-Knowledge protocols and applications to, e.g.
  - **Non-Transferability** of authentication: How to prove my identity without leaving a verifiable trace?
  - **Anonymity** in authentication: How to prove I belong to a group without revealing my identity?
  - **Catching Misbehaviour in General Protocols:** How to detect that a user doesn't follow a protocol?
- **Privacy in computation** (next lecture): general secure computation without a trusted party:, e.g.
  - Private e-voting
  - Private e-auctions
  - Private data mining...

Plan for this lecture

Zero-Knowledge (ZK) Proofs and Applications:

- Example Motivation: identification without a verifiable trace
- First example of a ZK Proof: Schnorr's protocol for proving knowledge of a DL secret
  - basic properties: completeness, soundness
  - new property: zero-knowledge – based on simulation
  - Second example: GQ proofs for RSA secret
- Generalization: ZK Proofs of Knowledge / Membership for any relation
  - Definition
  - Theoretical result: ZK protocol for any NP relation
  - Practical result: Sigma Protocols and Combining proofs via AND/OR
- Example applications (also, tutorial): anonymous authentication/credentials, catching protocol misbehaviour.

Example Motivation: identification without a verifiable trace

How to identify yourself with 'what you have'?

- Challenge-Response identification (ID) protocol?

Lots of distributed verifiers: don't want to store secret symmetric key in each verifier

- Digital signature-based challenge-Response ID protocol?

But... each identification leaves a verifiable signature trace behind!
Q.(Prover Privacy): How to avoid traceability, but still ensure impersonation unforgeability?
Possible A.: Use a Zero-Knowledge (ZK) Identification Protocol!

First example of a ZK Proof: Schnorr's DL protocol

Setup of Schnorr's ZK ID protocol (1991):

- Works in a cyclic group $G =< g >$ where Discrete-Logarithm (DL) problem is hard
- Fixed public generator $g \in G$ for G
- Denote order (size) of G by n (assumed prime).
  - e.g. (as in DSA digital signature standard): G a mutliplicative subgroup of $\mathbb{Z}_p^*$ (multiplicative group modulo p) for a prime p, where G is generated by $g \in \mathbb{Z}_p^*$, an element of prime order n, where n divides $p - 1$.
- Prover's Discrete-Log secret key: $x \hookleftarrow U(\mathbb{Z}_q)$.
- Prover's public-key: $h = g^x \in G$.
- For security parameter k (security level 2^k), ID protocol runs in k iterations.

First example of a ZK Proof: Schnorr's DL protocol

Proof of Knowledge of Discrete-Log: Prover has secret $x \in \mathbb{Z}_q$, Verifier has public $h = g^x \in G$

One iteration of Schnorr's ZK ID protocol (1991):

Prover		Verifier
$(x = \log_g h)$		
$u \in_R \mathbb{Z}_n$		
$a \leftarrow g^u$	$\xrightarrow{\ a\ }$	
		$c \in_R \{0,1\}$
	$\xleftarrow{\ c\ }$	
$r \leftarrow_n \begin{cases} u, & \text{if } c = 0 \\ u + x, & \text{if } c = 1 \end{cases}$	$\xrightarrow{\ r\ }$	$g^r \stackrel{?}{=} \begin{cases} a, & \text{if } c = 0 \\ ah, & \text{if } c = 1 \end{cases}$

FIGURE 4.2: Schnorr's zero-knowledge protocol

First example of a ZK Proof: Properties

Q: Why it a convincing 'proof of knowledge' of DL x for the verifier V?

A: Two reasons –

- Completeness: If P knows x, and P and V follow protocol, V's test will always pass.
- Soundness (informal statement): If P does not know x, and V follows protocol, V's test will pass with probability $\leq 1/2$.

Then, for full protocol (k iterations):

- if P knows x, V accepts with prob. 1, if P doesn't know x, V accepts with prob. $\leq 1/2^k$.

First example of a ZK Proof: Soundness

Q: Why does soundness hold for Schnorr's protocol? (intuition)
A: Suppose P doesn't know x, but guesses V's challenge c before sending commitment a:

- If P guesses $c = 0$, P prepares commitment $a = g^u$. If guess is right, respond to challenge with $r = u$.
- If P guesses $c = 1$, P prepares commitment $a = g^r h^{-1}$ for $r \hookleftarrow U(\mathbb{Z}_q)$. If guess is right, respond to challenge with r.

In both methods of choosing a, if P doesn't 'know' x, P can only respond to V's challenge correctly if it guessed c correctly!

- So, P's success probability $\leq 1/2$.

First example of a ZK Proof: Soundness Intuition (cont.)

Q: But why does P have to know x to respond correctly in both cases?

A: Suppose P somehow efficiently chooses a such that it can answer correctly to challenge in both cases $c = 0$ or $c = 1$: Then P knows $r_1, r_2 \in \mathbb{Z}_q$ such that:

$$g^{r_1} = a \text{ and } g^{r_2} = a \cdot h$$

Divide these equations: $g^{r_2 - r_1} = h$, so we can use P to efficiently compute $r_2 - r_1 = x$!

Conclusion: If P can respond correctly with success probability $> 1/2$, we can use P to efficiently compute the DL x.

- This latter is what we really mean by 'P knows x'
- Leads to formal definition of soundness based on 'know' $\equiv$ 'can efficiently compute').

First example of a ZK Proof: Zero Knowledge Property

Soundness is about security against an adversary prover.

Q: What can a curious verifier learn about x? (intuition)

A: Nothing it already doesn't know – zero knowledge property!

Why? Because there is an efficient simulator algorithm that V can use to simulate protocol messages (a, c, r) by itself, using just the public key $h = g^x$:

Real conversations

Input: private key x

Output: conversation $(a; c; r)$

1. $u \in_R \mathbb{Z}_n$
2. $a \leftarrow g^u$
3. $c \in_R \{0, 1\}$
4. $r \leftarrow_n u + cx$
5. output $(a; c; r)$

Simulated conversations

Input: public key h

Output: conversation $(a; c; r)$

1. $c \in_R \{0, 1\}$
2. $r \in_R \mathbb{Z}_n$
3. $a \leftarrow g^r h^{-c}$
4. output $(a; c; r)$

Both algorithms (left: real, right: sim) generate same distribution of triples (a, c, r): uniformly random such that $g^r = a \cdot h^c$.

First example of a ZK Proof: Zero Knowledge Property

Soundness is about security against an adversary prover.

Q: What can a curious verifier learn about x?

A: Nothing it already doesn't know – zero knowledge property!

Why? Because there is an efficient simulator algorithm that V can use to simulate protocol messages (a, c, r) by itself, using just the public key $h = g^x$:

Real conversations
Input: private key x
Output: conversation $(a; c; r)$

1. $u \in_R \mathbb{Z}_n$
2. $a \leftarrow g^u$
3. $c \in_R \{0, 1\}$
4. $r \leftarrow_n u + cx$
5. output $(a; c; r)$

Simulated conversations
Input: public key h
Output: conversation $(a; c; r)$

1. $c \in_R \{0, 1\}$
2. $r \in_R \mathbb{Z}_n$
3. $a \leftarrow g^r h^{-c}$
4. output $(a; c; r)$

Both algorithms (left: real, right: sim) generate same distribution of triples (a, c, r): uniformly random such that $g^r = a \cdot h^c$.

First example of a ZK Proof: Zero Knowledge Property

Previous simulation works for an honest but curious verifier V (follows protocol – picks c at random) – honest verifier ZK.

Q: What about a malicious verifier V^* that may not follow protocol (biased c)?

A: Still, nothing it already doesn't know – full zero knowledge!

Why? There is still an efficient simulator algorithm:

Real conversations
Input: private key x
Output: conversation $(a; c; r)$

1. $u \in_R \mathbb{Z}_n$
2. $a \leftarrow g^u$
3. send a to $\mathcal{V}^*$
4. receive $c \in \{0, 1\}$ from $\mathcal{V}^*$
5. $r \leftarrow_n u + cx$
6. send r to $\mathcal{V}^*$
7. output $(a; c; r)$

Simulated conversations
Input: public key h
Output: conversation $(a; c; r)$

1. $c \in_R \{0, 1\}$
2. $r \in_R \mathbb{Z}_n$
3. $a \leftarrow g^r h^{-c}$
4. send a to $\mathcal{V}^*$
5. receive $c' \in \{0, 1\}$ from $\mathcal{V}^*$
6. if $c \neq c'$ rewind $\mathcal{V}^*$ to point prior to receiving a and go to step 1
7. send r to $\mathcal{V}^*$
8. output $(a; c; r)$

Both algorithms (left: real, right: sim) generate same distribution of triples (a, c, r). Simulator still efficient: step 6 will be executed on average 2 times ($c = c'$ with prob. $1/2$).

Schnorr ZK Proof: Efficiency Improvement

Efficiency issue: repeat basic iteration k times for security 2^k.

Q: How to reduce to just one iteration?

A: Use exponentially large challenge space.

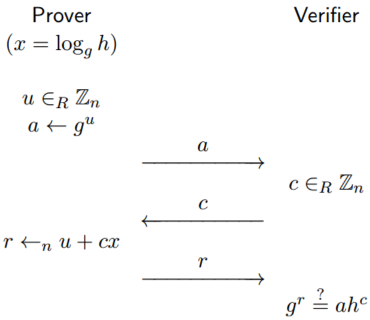


FIGURE 4.3: Schnorr's identification protocol

Drawback: Still honest verifier ZK, but lose provable full ZK property...

Another example ZK Proof: GQ – Proving knowledge of RSA decryption

GQ RSA-based ZK identification Protocol

Prover		Verifier
$(x = y^{1/e} \bmod m)$		
$u \in_R \mathbb{Z}_m^*$ $a \leftarrow_m u^e$		
	$\xrightarrow{a}$	
		$c \in_R \mathbb{Z}_e$
	$\xleftarrow{c}$	
$r \leftarrow_m ux^c$		
	$\xrightarrow{r}$	
		$r^e \stackrel{?}{=}_m ay^c$

Generalization: ZK Proofs for a relation

ZK is useful tool for proving something about a secret is true while minimizing leakage of info. on secret

Since discovery ([GMR85]), ZK has been extensively investigated and generalized to cover almost any imaginable scenario!

For instance, how to prove in ZK that:

- Anonymous authentication: I know a secret key that corresponds to one of N public keys of a group, without identifying which one.
- Anonymous credentials: I know a signature from an authority on my driver's licence (containing my name, address, age,...) but I just want to prove to an alcohol merchant that I am over 18, without leaking who I am.

To handle such general situations, need to generalize definition (and construction!) of ZK

Generalization: ZK Proofs for a relation

Generalizing the definition of ZK to any relation R:

- Let $R = \{(v; w)\} \subseteq V \times W)$ be a relation (e.g. $R = \{(v = (g, h); w = x) : h = g^x\}$ in Schnorr).
- Let $v \in V$ is the common public input to P and V (e.g. $h \in < g >$ in Schnorr)
- Let $w \in W$ is a witness private input to P (e.g. x such that $h = g^x$ in Schnorr).
- Let L_R be language corresponding to R (in theoret. Comp. Sci. terminology), i.e. set of $v \in V$ for which there exists a witness $w \in W$ with $(v; w) \in R$. (e.g. set $< g >$ in Schnorr)

Goal: For a given relation R, prove given v in ZK that I know a witness w such that $(v; w) \in R$.

Generalization: ZK Proofs for a relation

Generalizing the definition of ZK to any relation R (cont.)

The generalized desired properties:

- Completeness: If P and V follow protocol, V's test will always pass.
- Soundness: There exists an efficient (probabilistic polynomial time) algorithm (witness extractor) that given any malicious prover P^* that passes with non-negligible probability the honest verifier's test on input v, can extract a witness w such that $(v; w) \in R$.
- Zero Knowledge: The exists an efficient (expected polynomial time) algorithm (simulator) that given any malicious verifier V^*, can simulate protocol messages received by V^* from P on input v with a computationally indistinguishable distribution.

Generalization: ZK Proofs for a relation

Generalizing the construction of ZK to any relation R: Recall: A relation R is called an NP relation if R can be efficiently verified: given $(v; w)$ there is a polynomial time algorithm to decide if $(v; w) \in R$ or not. (basically all relations of practical interest!).

General theoretical result: Any effciently verifiable relation can also be proved in ZK!

Theorem [GMW86]: Any NP relation R has a polynomial time ZK proof protocol (using a collision-resistant hash function).

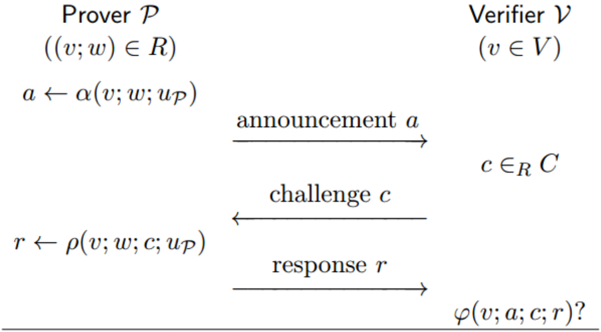
Practical issue: complexity of protocol is proportional to size of R's verification circuit. Tends to be impractical for most R. But shows generality of ZK in principle!

Idea (will not go through details):

- Give a ZK proof for Graph 3-Colourability (G3C) relation (NP-complete problem).
- Any NP relation R can be reduced to a Graph 3-Colourability (G3C) relation (by NP-completeness of G3C).
- To prove $(v; w) \in R$, apply reduction to get $(v'; w')$ and prove $(v'; w') \in G3C$. (the reduction can also efficiently transform w to w').

Practical result: Combining Sigma Protocols

More practical approach for many applications: generalize the Schnorr/GQ 'Sigma' type DL-based protocols



Conversation $(a; c; r)$ accepting if $\varphi(v; a; c; r)$ holds.
Polynomial time predicate φ, finite set $C \neq \emptyset$,
random tape $u_{\mathcal{P}}$, p.p.t. algorithms α and ρ.

FIGURE 5.1: Σ-protocol for relation R

Practical result: Combining Sigma Protocols

Idea: show how to combine Sigma protocols for existing relations to implement logical operators, such as:

- OR: Given 'Sigma' protocols for relations R_1, R_2, build a Sigma protocol for relation $R_1 \vee R_2 = \{(v_1, v_2; w_1, w_2) : (v_1; w_1) \in R_1 \vee (v_2; w_2) \in R_2\}$.
  - e.g. Anonymous identification: prove, given h_1, h_2, that I know x with $g^x = h_1$ or $g^x = h_2$.
- AND: Given 'Sigma' protocols for relations R_1, R_2, build a Sigma protocol for relation $R_1 \wedge R_2 = \{(v_1, v_2; w_1, w_2) : (v_1; w_1) \in R_1 \wedge (v_2; w_2) \in R_2\}$.
- EQ: Given 'Sigma' protocols for relations R_1, R_2, build a Sigma protocol for relation $R_1 \wedge R_2 = \{(v_1, v_2; w) : (v_1; w) \in R_1 \wedge (v_2; w) \in R_2\}$ – variant of 'AND' but prove witness used in both relation is same.
  - e.g.: Given $v_1 = (g_1, h_1), v_2 = (g_2, h_2)$, I know x with $g_1^x = h_1$ and $g_2^x = h_2$.

Practical result Example: OR Combination of Sigma Protocols

Idea: Split challenge into a sum of two subchallenges (prover can 'cheat' in at most one of them)

Prover			Verifier
(using $x_1 = \log_g h_1$)	(using $x_2 = \log_g h_2$)		
$c_2, r_2, u_1 \in_R \mathbb{Z}_n$	$c_1, r_1, u_2 \in_R \mathbb{Z}_n$		
$a_1 \leftarrow g^{u_1}$	$a_1 \leftarrow g^{r_1} h_1^{-c_1}$		
$a_2 \leftarrow g^{r_2} h_2^{-c_2}$	$a_2 \leftarrow g^{u_2}$	$\xrightarrow{a_1, a_2}$	
			$c \in_R \mathbb{Z}_n$
		$\xleftarrow{c}$	
$c_1 \leftarrow_n c - c_2$	$c_2 \leftarrow_n c - c_1$		
$r_1 \leftarrow_n u_1 + c_1 x_1$	$r_2 \leftarrow_n u_2 + c_2 x_2$	$\xrightarrow{c_1, c_2, r_1, r_2}$	$c_1 + c_2 \stackrel{?}{=}_n c$
			$g^{r_1} \stackrel{?}{=} a_1 h_1^{c_1}$
			$g^{r_2} \stackrel{?}{=} a_2 h_2^{c_2}$

FIGURE 5.8: OR-composition of Schnorr's protocol

Practical result Example: EQ Combination of Sigma Protocols

Idea: Use same challenge and response for both relations

Prover		Verifier
$(x = \log_{g_1} h_1 = \log_{g_2} h_2)$		
$u \in_R \mathbb{Z}_n$		
$a_1 \leftarrow g_1^u$		
$a_2 \leftarrow g_2^u$	$\xrightarrow{a_1, a_2}$	
		$c \in_R \mathbb{Z}_n$
	$\xleftarrow{c}$	
$r \leftarrow_n u + cx$		
	$\xrightarrow{r}$	
		$g_1^r \stackrel{?}{=} a_1 h_1^c$
		$g_2^r \stackrel{?}{=} a_2 h_2^c$

FIGURE 5.7: EQ-composition of Schnorr's protocol

Example Applications of ZK Proofs

General Application in Crypto. Protocols: Check parties are following the protocol, without leaking info:

- Suppose protocol P designed to be secure only against honest but curious attacks.
  - But P insecure against malicious parties not following protocol
- To strengthen P into P' secure against malicious parties, idea:
  - Whenever P specifies party n sends $z = f(x, y)$ ($x =$ party's secret, y=other protocol messages), in protocol P', party n sends $z = f(x, y)$ and ZK proof π that P knows x such that $z = f(x, y)$.
  - Receivers verify the proof; if ver. fails, stop protocol and remove malicious party P.

Example Applications of ZK Proofs

Anonymous authentication applications (basic ideas, tute for more):

Anonymous, offline electronic cash (Chaum et al, 1990s):

- **Goals:**
  - Anonymous payment,
  - unlinkable payments by same identity,
  - avoid online 'double-spending' check
- **Techniques:**
  - 'blind' signatures for anonymity/unlinkability (signer doesn't see coin being signed), but
  - payment reveals to merchant a function of customer identity
  - Two 'double spending' payments on same coin will reveal full identity! (offline).
  - Critical Role of ZK: force customer to reveal function of its identity (see prev. slide)!

Example Applications of ZK Proofs

Anonymous authentication applications (basic ideas, tute for more):

Anonymous credentials (Brand 1990s, Camenisch/Lysyanskaya 2000's): Signed credentials (e.g. driver's licence) with multiple attributes by authority

- **Goals:**
  - Selective disclosure of attributes when showing credentials
  - unlinkability between showing and issuing sessions
- **Techniques:**
  - credential = authority signature on function (commitment) of attributes
  - Showing credentials: ZK proof that: have a signauture on function of attributes, commitment matches revealed attributes.

Example Applications of ZK Proofs

Group Signatures: Anyone can sign on behalf of N-signer group

- **Goals:**
  - Anonymity/Unlinkability: Identity of signer in group and linking its signatures should be hard
  - Revoking Anonymity: A group manager can revoke (open) anonymity to determine who produces each signature (e.g. in disputes/fraud).
  - Unframeability: Group members / Group Manager should not be able to frame an innocent group member.
- **Techniques:**
  - signature = proof of knowledge of secret key for 1-of-N public keys (N-wise OR proof).
  - For opening: include in signature encryption of signer's public key under group manager's public key
  - To prevent framing by users: include in signature ZK proof that encryption encrypts key used for signing!
  - To prevent framing by Group manager: prove in ZK that decryption was correctly done!