Practical Zero-Knowledge Proofs for Circuit Evaluation

E. Ghadafi N.P. Smart B. Warinschi

Department of Computer Science,
University of Bristol

Twelfth IMA International Conference on Cryptography and Coding $15^{th} - 17^{th}$ December 2009

OUTLINE

OUTLINE

Outline

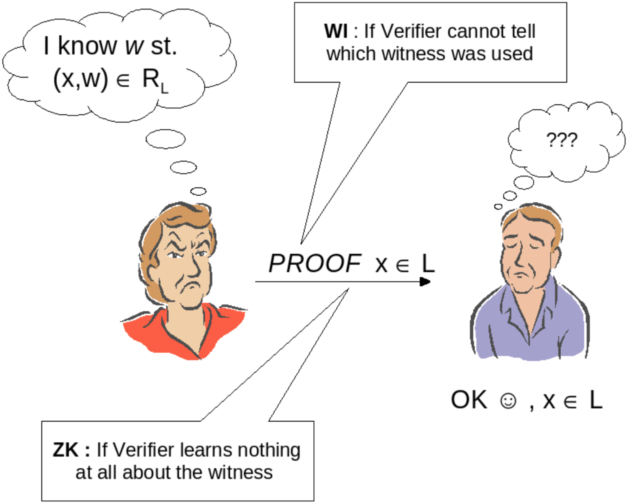
OUTLINE

Outline

OUTLINE

NON-INTERACTIVE PROOFS

"A proof is whatever convinces me.", Shimon Even.



APPLICATIONS OF ZERO-KNOWLEDGE PROOFS

Example applications:

- **Anonymous Credentials:** Client proves he possesses the required credentials without revealing them.
- **Online Voting:** Voter proves to the server that he has voted correctly without revealing his actual vote.
- **Signature Schemes, Oblivious Transfer , CCA-2 Encryption Schemes, ...**

HISTORY OF NIZK PROOFS

- Blum-Feldman-Micali, 1988.
- Damgard, 1992.
- Killian-Petrank, 1998.
- Feige-Lapidot-Shamir, 1999.
- De Santis-Di Crescenzo-Persiano, 2002.
- Groth-Sahai, 2008.

OUR CONTRIBUTION

- Efficient implementations of NIZK proofs for Circuit SAT in the ROM model using Sigma-Protocols and other optimizations (e.g. Computing shared monomials, etc.).
- Efficient implementations of NIZK proofs for Circuit SAT in the CRS model using Groth-Sahai proofs.

Implementation (Rationale)

Why Circuits ???

- Every *NP* problem could be reduced to Circuit SAT.
  Problem: Circuit Size ???
  Solution: Efficient implementations would help solve some of this problem.
- Other techniques that does not require reduction to *NP* are applicable to limited languages (i.e. You cannot prove much with them).

ROM PROOFS-Σ PROTOCOLS

Prover

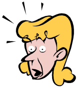
Public Parameters,

(w, x)

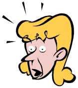
Verifier

Public Parameters,

(x)



ROM PROOFS-Σ PROTOCOLS

Prover		**Verifier**
Public Parameters, (w, x)		Public Parameters, (x)
	$\xrightarrow{\text{Commitment}}$	

ROM PROOFS-Σ PROTOCOLS

Prover

Public Parameters,

(w, x)

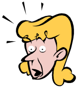
Commitment $\longrightarrow$

Challenge $\longleftarrow$

Verifier

Public Parameters,

(x)



ROM Proofs-Σ Protocols

Prover

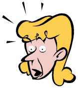
Public Parameters,

(w, x)

Verifier

Public Parameters,

(x)



$$\xrightarrow{\text{Commitment}}$$

$$\xleftarrow{\text{Challenge}}$$

$$\xrightarrow{\text{Response}}$$

ROM PROOFS-Σ PROTOCOLS

Prover

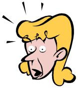
Public Parameters,

(w, x)

Verifier

Public Parameters,

(x)



Commitment $\longrightarrow$

Challenge $\longleftarrow$

Response $\longrightarrow$

$\Downarrow$

Accept or Reject

ROM PROOFS-Σ PROTOCOLS

Prover

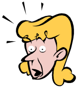
Public Parameters,

(w, x)

Verifier

Public Parameters,

(x)



$\xrightarrow{\text{Commitment}}$

$\xleftarrow{\text{Challenge}}$

$\xrightarrow{\text{Response}}$

$\Downarrow$

Accept or Reject

- The interactive proof could be made non-interactive using the Fiat-Shamir transformation. The challenge is now: H(Public parameters || Commitment)

Groth-Sahai Proofs

Symmetric External Diffie-Hellman Assumption Proofs:

Setup:

$$\mathbb{A}_1 \quad \times \quad \mathbb{A}_2 \quad \xrightarrow{f} \quad \mathbb{A}_T$$

Groth-Sahai Proofs

Symmetric External Diffie-Hellman Assumption Proofs:

Setup:

$$
\begin{array}{ccccc}
\mathbb{A}_1 & \times & \mathbb{A}_2 & \xrightarrow{f} & \mathbb{A}_T \\
\mathbb{B}_1 & \times & \mathbb{B}_2 & \xrightarrow{F} & \mathbb{B}_T
\end{array}
$$

GROTH-SAHAI PROOFS

Symmetric External Diffie-Hellman Assumption Proofs:

Setup:

$$\begin{array}{ccccc} \mathbb{A}_1 & \times & \mathbb{A}_2 & \xrightarrow{f} & \mathbb{A}_T \\ \iota_1 \downarrow\uparrow \rho_1 & & \iota_2 \downarrow\uparrow \rho_2 & & \iota_T \downarrow\uparrow \rho_T \\ \mathbb{B}_1 & \times & \mathbb{B}_2 & \xrightarrow{F} & \mathbb{B}_T \end{array}$$

Properties:

$$\forall x \in \mathbb{A}_1, \forall y \in \mathbb{A}_2 : F(\iota_1(x), \iota_2(y)) = \iota_T(f(x,y)),$$
$$\forall \mathcal{X} \in \mathbb{B}_1, \forall \mathcal{Y} \in \mathbb{B}_2 : f(p_1(\mathcal{X}), p_2(\mathcal{Y})) = p_T(F(\mathcal{X}, \mathcal{Y})).$$

Proof:

Consists of $\Theta \in \mathbb{B}_1$ and $\Pi \in \mathbb{B}_2$

Groth-Sahai Proofs

- **Product Proof:** Prove that one value is the product of other two values.

  Equation: $\vec{x_1}^{(1)} \cdot \vec{x_2}^{(1)} - \vec{x_1}^{(2)} = 0.$

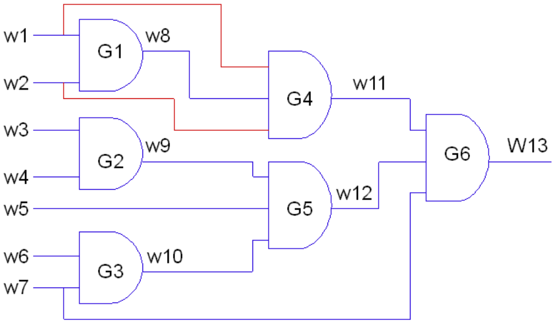
- **Bit Proof:** Prove that a commitment hides 0 or 1.

  Equation: $\vec{x_1}^{(1)} \cdot \vec{x_2}^{(1)} - \vec{x_1}^{(1)} = 0.$

- **Equality Proof:** Prove that two different commitments hide the same value.

  Equation: $\vec{x_2}^{(1)} - \vec{x_1}^{(1)} = 0.$

IMPLEMENTATION



$\mathcal{I}$	:	The circuit input wires $\{w_1, ..., w_7\}$
$\mathcal{O}$	:	The circuit final output wires $\{w_{13}\}$
$\mathcal{G}$	:	The set of gates $\{g_1, ..., g_6\}$
$\mathcal{M}on$	:	The set of monomials (i.e. products needed in the QEq Method)
$\mathcal{PW}$	:	The set of proof wires (i.e. wires shared between monomials)

LEQ-METHOD

- **LEq Method (Groth et al.):**
  Each gate is represented by linear equation as follows :

  $$out = a \cdot x + b \cdot y + c \cdot z + d, \text{ where } out \in \{0, 1\}$$

  For each 2-to-1 gate, there exists unique values for a,b,c and d that makes the above equation hold.
  OR gate as an example: we have $a = -1$,$b = -1$, $c = 2$ and $d = 0$.

x	y	**z**	out	other
0	0	**0**	0	2
0	1	**1**	1	-1
1	0	**1**	1	-1
1	1	**1**	0	-2

Implementation of LEq-Method

Prover for LEq-Method

- Evaluate every wire in the circuit given the input.

Implementation of LEq-Method

Prover for LEq-Method

- Evaluate every wire in the circuit given the input.
- $\forall w_i \in \mathcal{W}$ compute $comm_i = comm(w_i, r_i)$.

Implementation of LEq-Method

Prover for LEq-Method

- Evaluate every wire in the circuit given the input.
- $\forall w_i \in \mathcal{W}$ compute $comm_i = comm(w_i, r_i)$.
- $\forall i \in \mathcal{W}$, Prove $comm_i \in \{0, 1\}$.

Implementation of LEq-Method

Prover for LEq-Method

- Evaluate every wire in the circuit given the input.
- $\forall w_i \in \mathcal{W}$ compute $comm_i = comm(w_i, r_i)$.
- $\forall i \in \mathcal{W}$, Prove $comm_i \in \{0, 1\}$.
- $\forall i \in \mathcal{G}$, prove that the linear equation value $\in \{0, 1\}$.

Implementation of LEq-Method

Prover for LEq-Method

- Evaluate every wire in the circuit given the input.
- $\forall w_i \in \mathcal{W}$ compute $comm_i = comm(w_i, r_i)$.
- $\forall i \in \mathcal{W}$, Prove $comm_i \in \{0, 1\}$.
- $\forall i \in \mathcal{G}$, prove that the linear equation value $\in \{0, 1\}$.
- Output the decommitment(i.e. Wire values and the randomness used in the commitment) of the circuit's final output wires(i.e. the set $\mathcal{O}$).

Implementation of LEq-Method

Verifier for LEq-Method

- For all wires, verify that $comm_i \in \{0, 1\}$.

Implementation of LEq-Method

Verifier for LEq-Method

- For all wires, verify that $comm_i \in \{0, 1\}$.
- For each gate, verify that the linear equation value $\in \{0, 1\}$.

IMPLEMENTATION OF LEQ-METHOD

VERIFIER FOR LEQ-METHOD

- For all wires, verify that $comm_i \in \{0, 1\}$.
- For each gate, verify that the linear equation value $\in \{0, 1\}$.
- For each gate, verify that the linear equation was formed correctly.

IMPLEMENTATION OF LEQ-METHOD

VERIFIER FOR LEQ-METHOD

- For all wires, verify that $comm_i \in \{0, 1\}$.
- For each gate, verify that the linear equation value $\in \{0, 1\}$.
- For each gate, verify that the linear equation was formed correctly.
- Compare the final output commitments of the circuit with those of the prover and Accept if they are identical, or Reject otherwise.

QEq-Method

- **QEq Method:**

  Each gate is represented by a quadratic equation as follows:

  $$z = a_0 + a_1 \cdot y + a_2 \cdot x + a_3 \cdot x \cdot y$$

  OR gate as an example :

x	y	z	
0	0	**0**	$\Leftarrow z_0$
0	1	**1**	$\Leftarrow z_1$
1	0	**1**	$\Leftarrow z_2$
1	1	**1**	$\Leftarrow z_3$

$$\begin{aligned} a_0 &= z_0 \\ a_1 &= z_1 - a_0 \\ a_2 &= z_2 - a_0 \\ a_3 &= z_3 - a_0 - a_1 - a_2 \end{aligned}$$

Implementation of QEq-Method

Prover for QEq-Method

- Evaluate the circuit given the input.

IMPLEMENTATION OF QEQ-METHOD

PROVER FOR QEQ-METHOD

- Evaluate the circuit given the input.
- Compute a commitment to each input wire $comm_i = comm(w_i, r_i)$ where $w_i \in \mathcal{I}$.

Implementation of QEq-Method

Prover for QEq-Method

- Evaluate the circuit given the input.
- Compute a commitment to each input wire $comm_i = comm(w_i, r_i)$ where $w_i \in \mathcal{I}$.
- Generate a proof that $comm_i$ will open to an element $\in \{0, 1\}$ for $i = 1, ..., |\mathcal{I}|$.

IMPLEMENTATION OF QEQ-METHOD

PROVER FOR QEQ-METHOD

- Evaluate the circuit given the input.
- Compute a commitment to each input wire $comm_i = comm(w_i, r_i)$ where $w_i \in \mathcal{I}$.
- Generate a proof that $comm_i$ will open to an element $\in \{0, 1\}$ for $i = 1, ..., |\mathcal{I}|$.
- For every element of $\mathcal{M}on$, compute a commitment to the product $comm_{i,j} = comm(w_i * w_j, r_{i,j})$.

Implementation of QEq-Method

Prover for QEq-Method

- Evaluate the circuit given the input.
- Compute a commitment to each input wire $comm_i = comm(w_i, r_i)$ where $w_i \in \mathcal{I}$.
- Generate a proof that $comm_i$ will open to an element $\in \{0, 1\}$ for $i = 1, ..., |\mathcal{I}|$.
- For every element of $\mathcal{M}on$, compute a commitment to the product $comm_{i,j} = comm(w_i * w_j, r_{i,j})$.
- For each gate ,g_i, compute a commitment $comm_k$ of the output wire w_k via $comm(w_k, r_k) = comm_{a_0} + a_2 \cdot comm_i + a_1 \cdot comm_j + a_3 \cdot comm_{i*j}$

IMPLEMENTATION OF QEQ-METHOD

PROVER FOR QEQ-METHOD

- Evaluate the circuit given the input.
- Compute a commitment to each input wire $comm_i = comm(w_i, r_i)$ where $w_i \in \mathcal{I}$.
- Generate a proof that $comm_i$ will open to an element $\in \{0, 1\}$ for $i = 1, ..., |\mathcal{I}|$.
- For every element of $\mathcal{M}on$, compute a commitment to the product $comm_{i,j} = comm(w_i * w_j, r_{i,j})$.
- For each gate ,g_i, compute a commitment $comm_k$ of the output wire w_k via $comm(w_k, r_k) = comm_{a_0} + a_2 \cdot comm_i + a_1 \cdot comm_j + a_3 \cdot comm_{i*j}$
- For all monomials, generate a proof that the commitments $comm_{i*j}$ are consistent with the wire commitments(i.e. do product proofs together).

IMPLEMENTATION OF QEQ-METHOD

PROVER FOR QEQ-METHOD

- Evaluate the circuit given the input.
- Compute a commitment to each input wire $comm_i = comm(w_i, r_i)$ where $w_i \in \mathcal{I}$.
- Generate a proof that $comm_i$ will open to an element $\in \{0, 1\}$ for $i = 1, ..., |\mathcal{I}|$.
- For every element of $\mathcal{M}on$, compute a commitment to the product $comm_{i,j} = comm(w_i * w_j, r_{i,j})$.
- For each gate ,g_i, compute a commitment $comm_k$ of the output wire w_k via $comm(w_k, r_k) = comm_{a_0} + a_2 \cdot comm_i + a_1 \cdot comm_j + a_3 \cdot comm_{i*j}$
- For all monomials, generate a proof that the commitments $comm_{i*j}$ are consistent with the wire commitments(i.e. do product proofs together).
- Output the decommitment values of the final output wires.

Implementation of QEq-Method

Verifier for QEq-Method

- $\forall i \in \mathcal{I}$, verify that $comm_i$ will open to an element $\in \{0, 1\}$.

Implementation of QEq-Method

Verifier for QEq-Method

- $\forall i \in \mathcal{I}$, verify that $comm_i$ will open to an element $\in \{0, 1\}$.
- Compute the rest of wires' commitments (Taking advantage of the homomorphic property of the commitment scheme).

Implementation of QEq-Method

Verifier for QEq-Method

- $\forall i \in \mathcal{I}$, verify that $comm_i$ will open to an element $\in \{0, 1\}$.
- Compute the rest of wires' commitments (Taking advantage of the homomorphic property of the commitment scheme).
- Verify all product proofs .

Implementation of QEq-Method

Verifier for QEq-Method

- $\forall i \in \mathcal{I}$, verify that $comm_i$ will open to an element $\in \{0, 1\}$.
- Compute the rest of wires' commitments (Taking advantage of the homomorphic property of the commitment scheme).
- Verify all product proofs .
- Compare the final output commitments of the circuit with those of the prover and Accept if they are identical, or Reject otherwise.

BATCH VERIFICATION

Motivation:

Verification of individual proofs takes a lot of time, so we use batch verification to save some time.

Batch Verification

Motivation:

Verification of individual proofs takes a lot of time, so we use batch verification to save some time.

Batch verification in the ROM model:

- Small Exponent Test(Bellare et al.):
  To check that $y_1 = g^{x_1}, \ldots, y_n = g^{x_n}$

Batch Verification

Motivation:

Verification of individual proofs takes a lot of time, so we use batch verification to save some time.

Batch verification in the ROM model:

- Small Exponent Test(Bellare et al.):
  To check that $y_1 = g^{x_1}, \ldots, y_n = g^{x_n}$
  - Choose $\gamma_1, \ldots, \gamma_n$ at random where $|\gamma_i| = l$.
  - Compute $X = \sum_{i=1}^{n} (x_i \cdot \gamma_i)$ and $Y = \prod_{i=1}^{n} y_i^{\gamma_i}$.
  - The verification is done by checking that $g^X = Y$.
- There are different ways to efficiently compute product of powers(i.e. Y).

Batch Verification

Batch verification in the CRS model:

- **Product Proof:** To verify a single Product Proof, one checks:

$$F\left(\vec{C_1}^{(2)}, -\mathcal{W}_2\right) \cdot F\left(\vec{C_1}^{(1)}, \vec{C_2}^{(1)}\right) \cdot F(-\mathcal{U}_1, \Pi) \cdot F(\Theta, -\mathcal{U}_2) = 1$$

Batch Verification

Batch verification in the CRS model:

- **Product Proof:** To verify a single Product Proof, one checks:

$$F\left(\vec{C_1}^{(2)}, -\mathcal{W}_2\right) \cdot F\left(\vec{C_1}^{(1)}, \vec{C_2}^{(1)}\right) \cdot F(-\mathcal{U}_1, \Pi) \cdot F(\Theta, -\mathcal{U}_2) = 1$$

Only need $n + 3$ products of *Four lots* of pairings compared to $4n$ products of *Four lots* of pairings.

Batch Verification

Batch verification in the CRS model:

- **Product Proof:** To verify a single Product Proof, one checks:

$$F\left(\vec{C_1}^{(2)}, -\mathcal{W}_2\right) \cdot F\left(\vec{C_1}^{(1)}, \vec{C_2}^{(1)}\right) \cdot F(-\mathcal{U}_1, \Pi) \cdot F(\Theta, -\mathcal{U}_2) = 1$$

  Only need $n + 3$ products of *Four lots* of pairings compared to $4n$ products of *Four lots* of pairings.

- **Bit Proof:** To verify a single Bit Proof, one checks:

$$F\left(\vec{C_1}^{(1)}, -\mathcal{W}_2\right) \cdot F\left(\vec{C_1}^{(1)}, \vec{C_2}^{(1)}\right) \cdot F(-\mathcal{U}_1, \Pi) \cdot F(\Theta, -\mathcal{U}_2) = 1$$

  Only need $n + 3$ products of *Four lots* of pairings compared to $4n$ products of *Four lots* of pairings.

BATCH VERIFICATION

Batch verification in the CRS model:

- **Product Proof:** To verify a single Product Proof, one checks:

$$F\left(\vec{C_1}^{(2)}, -\mathcal{W}_2\right) \cdot F\left(\vec{C_1}^{(1)}, \vec{C_2}^{(1)}\right) \cdot F(-\mathcal{U}_1, \Pi) \cdot F(\Theta, -\mathcal{U}_2) = 1$$

Only need $n+3$ products of *Four lots* of pairings compared to $4n$ products of *Four lots* of pairings.

- **Bit Proof:** To verify a single Bit Proof, one checks:

$$F\left(\vec{C_1}^{(1)}, -\mathcal{W}_2\right) \cdot F\left(\vec{C_1}^{(1)}, \vec{C_2}^{(1)}\right) \cdot F(-\mathcal{U}_1, \Pi) \cdot F(\Theta, -\mathcal{U}_2) = 1$$

Only need $n+3$ products of *Four lots* of pairings compared to $4n$ products of *Four lots* of pairings.

- **Equality Proof:** To verify a single Equality Proof, one checks:

$$F\left(\vec{C_1}^{(1)}, -\mathcal{W}_2\right) \cdot F\left(\mathcal{W}_1, \vec{C_2}^{(1)}\right) \cdot F(-\mathcal{U}_1, \Pi) \cdot F(\Theta, -\mathcal{U}_2) = 1$$

Only need 4 products of *Four lots* of pairings(16 pairings) compared to $4n$ products of *Four lots* of pairings($16n$ Pairings)!!!

Proof Sizes Comparison

Parameter	LEq-Method	QEq-Method
Commitments	$\lvert\mathcal{W}\rvert$	$\lvert\mathcal{I}\rvert + \lvert\mathcal{M}on\rvert$
Bit Proofs	$\lvert\mathcal{W}\rvert + \lvert\mathcal{G}\rvert$	$\lvert\mathcal{I}\rvert$
Product Proofs	-	$\lvert\mathcal{PW}\rvert^1 or \lvert\mathcal{M}on\rvert^2$
Decommitments	$\lvert\mathcal{O}\rvert$	$\lvert\mathcal{O}\rvert$

[1] If we are using the Random Oracle Model.

[2] If we are using the Common Reference String Model.

Circuits' Details

Circuit-1: 32-bit integers comparison.
Circuit-2: AES-128(Prove that the plain text was encrypted under the secret key).

Table: Details of the two circuits used in the experiments

Parameter	Circuit-1	Circuit-2
Gates	184	33880
Input Wires	64	128
Output Wires	1	128
Total Wires	248	34136
$\lvert\mathcal{PW}\rvert$	93	15596
$\lvert\mathcal{M}on\rvert$	154	32244

Curves Used

ROM: secp256r1 curve from the SECG standard.
CRS: 256–bit Barreto-Naehrig curve.

Results and Timings

All our timings are in seconds and were tested on a Linux machine with Intel Core Duo 3.00GHz processor.

Table: Timings for our two circuits

Model	Circuit	Proof Method	Prover Time	Verifier Time	Batch Time	Time Saved
ROM	1	LEq	4.7	5.3	1.97	62.8%
ROM	1	QEq	1.95/2.25	2.5	2.01/1.28	19.6%/48.8%
ROM	2	LEq	729	839	321	61.7%
ROM	2	QEq	296/280	372	360/253	3.2%/31.9%
CRS	1	LEq	44	450	64	85.8%
CRS	1	QEq	15.23	163	29.5	81.9%
CRS	2	LEq	7174	70300	9431	86.6%
CRS	2	QEq	2406	24861	4200	83.1%

SUMMARY

- QEq method is faster than the LEq method.
- Computing the shared monomials saves time.
- GS proofs are slower than the ROM proofs. This is no surprise as proofs in the standard model are usually less efficient than the ROM ones.
- GS proof verification is faster when using the "pairing product" trick.
- Batch verification is very beneficial in Groth-Sahai proofs.

The End

The End.

Questions?