Eurocrypt Cologne 2009

International Association for Cryptologic Research

Automatic Generation of Sound Zero-Knowledge Protocols

Endre Bangerter[1], Jan Camenisch[2], Stephan Krenn[1,3], Ahmad-Reza Sadeghi[4], Thomas Schneider[4]

[1] Bern University of Applied Sciences, Biel-Bienne, Switzerland
[2] IBM Research, Zurich Research Labs, Rüschlikon, Switzerland
[3] University of Fribourg, Switzerland
[4] System Security Group, Ruhr-University Bochum, Germany

Introduction

- **ZK Proofs are Basic Crypto Primitives**
  - Used in Identification Schemes, Group Signatures, Secure Multiparty Computation, ...
- **First ZK Protocols Deployed in Practice**
  - Direct Anonymous Attestation (DAA) in TPM chips
  - Anonymous Credential Systems (IBM identity mixer)
- **Based on Efficient Sigma-Protocols**
  - Protocol Composition (AND, OR, ...)
  - Generic Transformations to Non-Interactive ZK (NIZK), ...
- **Design & Implementation "by Hand"**
  - Time-Consuming, Error-Prone, ...

Challenges

- **Protocol Design**
  - Choose Suitable Proof Techniques and Parameters
- **Implementation Efficiency**
  - From Semantic Proof Goal to Implementation
  - Skill Gap between Cryptographer & Programmers
- **Code Efficiency & Security**
  - Optimize Resources at Protocol & Code Level
  - Buffer Overflows, SW Side-Channels, ...

Automatic Generation of Sound ZK Protocols

High Level ZK-PoK Language

Relation to Prove, Security Level, Optimization Constraints, ...

Protocol Specification Language

Multiple Types of Sigma-Protocols, Groups, Arbitrary Compositions (AND, OR, ...), ...

Protocol Implementation Language

Algorithms, Operations, Messages, ...

Semantic Goal — What to prove?

↓ High Level Compiler — Automatically find most efficient proof technique.

↓ Protocol Specification — How to prove?

↓ Protocol Compiler — Automatic protocol composition and choice of parameters. Plugins: NIZK, Costs

↓ Protocol Implementation — Details of proof protocol.

Multiple Backends: C Code, LaTeX Documentation, CAO Code

C Code: Easy to change from GMP to other libraries.

CAO Code: Cryptography Aware Language and Compiler: Implementation of cryptographic primitives w.r.t. performance & security.

```
ProtocolInputs{
  Declaration := Prime(1024) p; Prime(160) q;
                 G=Zmod+(q) x; H=Zmod*(p) g, y;
  ProverPrivate  := x;
  ProverPublic   := p,q,g,y;
  VerifierPublic := p,q,g,y;
}

KnowledgeError        := 80;
ProtocolComposition   := P_1 Or P_2;

Def SigmaPhi P_1 {
  ChallengeLength   := 80;
  Homomorphism (phi : G -> H : (a) |-> (g^a));
  Relation ((y) = phi(x));
}

Def SigmaExp P_2 {
  ModulusLength := 8192;
  Homomorphism ...
  Relation ...
}
```

Σ^{exp} – an Efficient Unconditionally Portable Protocol

Applications

Proofs in hidden-order groups:

- Interval proofs
  - Boudot (Eurocrypt 00)
  - Lipmaa (ePrint 01)
- Anonymous Credential Systems
  - Camenisch, Lysyanskaya (SCN 02)
  - Camenisch, Van Herreweghen (ACM CCS 02)
- E-Cash
  - Camenisch, Hohenberger, Lysyanskaya (Eurocrypt 05)

High-Level Description of Σ^{exp}

$$ZPK[(x_1,\dots,x_m): y=\phi(x_1,\dots,x_m)]$$

Auxiliary String

n : RSA-Modulus

$g \in \mathbb{Z}_n^*$

$g_1,\dots,g_m \in \langle g \rangle$

Prover

$t := \phi(r_1,\dots,r_m)$

$t' := g_1^{r_1}\dots g_m^{r_m} g^r$

$y' := g_1^{x_1}\dots g_m^{x_m} g^x$

Prover → Verifier: t, t', y'

Verifier

$c \in C$

Verifier → Prover: c

$s_i := r_i + c x_i$

$s := r + c x$

Prover → Verifier: $s_1,\dots,s_m,s$

$t y^c \stackrel{?}{=} \phi(s_1,\dots,s_m)$

$t' y'^c \stackrel{?}{=} g_1^{s_1}\dots g_m^{s_m} g^s$

Related Work

- Bangerter (PhD-Thesis 05)
- Bangerter, Camenisch, Maurer (PKC 05)
- Camenisch, Kiayias, Yung (Eurocrypt 09)

Advantage

Σ^{exp} has 3 instead of 6 moves:

- Higher Efficiency
- Efficient Transformations to
  - Non-Interactive ZK
  - Concurrent ZK
- Generic Protocol Composition

EU FP7 Project Computer Aided Cryptography Engineering (CACE) - http://www.cace-project.eu

cace