Forward-secrecy on POP

Arthur Villard

School of Computer and Communication Sciences
Decentralized and Distributed Systems lab
Master Thesis – September 2017

Responsible
Prof. Bryan Ford
EPFL / DEDIS

Supervisor
Prof. Ewa Syta
Trinity College

Linus Gasser
EPFL / DEDIS
Context

- Online collaborative service (e.g. Wikipedia)
- Authenticate users anonymously against a list
- Link authentication attempts
- Other example: e-voting
Overview

- Introduction
- PoP and DAGA interaction
- Implementing DAGA
- Improving DAGA
- Conclusion & Future work
Overview

- Introduction
- PoP and DAGA interaction
- Implementing DAGA
- Improving DAGA
- Conclusion & Future work
Frameworks

• **PoP**: Proof of Personhood – DEDIS
  - Creation of the user list
  - Authentication protocol
  - Anonymity within the group
  - No forward-secrecy

• **DAGA**: Deniable Anonymous Group Authentication – Ewa Syta
  - Authentication protocol
  - Forward-secrecy
Goals

- Using Daga as PoP’s authentication protocol
- Implementing Daga in Go
- Improving Daga
Key concepts

• **Anonymity**
  ➔ No information about the user is known

• **Accountability**
  ➔ The sender can be held responsible for his action

• **Linkability**
  ➔ Two messages come from the same user

• **Forward-secrecy**
  ➔ Breaking a session does not break the previous ones
Overview

- Introduction
- PoP and DAGA interaction
- Implementing DAGA
- Improving DAGA
- Conclusion & Future work
Integration

1. Context
2. Challenge generation
3. Request
4. Linkage tag & Proofs

POP

Service

Party transcript

DAGA servers

User

DAGA

12/02/2018
Arthur Villard - Master Thesis

9
PoP: Weaknesses

• No forward-secrecy
  → Tag derived from private key
  → Leakage allows to identify the user in previous sessions

• Cross-service de-anonymisation
  → Tags independent from the service
  → Users can be tracked between different services

→ Loss of anonymity
DAGA: How it works

Introduction | PoP and DAGA interaction | Implementing DAGA | Improving DAGA | Conclusion

User

Context

DAGA servers

Compute initial tag

Proof generation

Request generation

Request challenge

Challenge

Request

Linkage tag + Proofs

Distributed randomness

Request generation

Compute initial tag

Proof generation

Distributed randomness

User

Context

DAGA servers

Compute initial tag

Proof generation

Request generation

Request challenge

Challenge

Request

Linkage tag + Proofs

Distributed randomness

User

Context

DAGA servers

Compute initial tag

Proof generation

Request generation

Request challenge

Challenge

Request

Linkage tag + Proofs

Distributed randomness
DAGA solutions

• **Forward-secrecy**
  - Tags derived from context elements *only*
  - Private key used in client proof
  - Proof does not leak information

• **Cross-service de-anonymisation**
  - Different services → Different contexts
    - Different tags for the same user
Conclusion

- DAGA can solve PoP weaknesses
- DAGA and PoP can be interfaced
- E-voting
Overview

- Introduction
- PoP and DAGA interaction
- Implementing DAGA
- Improving DAGA
- Conclusion & Future work
Implementation

- Go
- RSA Elliptic Curves
- Distributed randomness

\[ T_0^i = h_i \left( \prod_{k=1}^{m} s_k \right) \]
\[ T_0^i = \left( \prod_{k=1}^{m} s_k \right) \ast H_i \]
Code results

- Library: Complete implementation
- Test coverage 88%
- Example scenario
- Benchmark package
Introduction | PoP and DAGA interaction | Implementing DAGA | Improving DAGA | Conclusion

Benchmarks: Communication

- No improvement
- No explanation yet
Benchmarks: Time

Setup:
- Ubuntu 12.04
- x86-64
- 1 thread

32768 members / 32 servers  15 000 s

/15

1 000 s

- Moore’s law 2012 → 2018: ~ /8 from hardware
- Elliptic Curves
Conclusion

- Complete implementation
- Time improvement
- Next step: Integrate it with PoP
Overview

- Introduction
- PoP and DAGA interaction
- Implementing DAGA
- Improving DAGA
- Conclusion & Future work
Proof problem

Introduction | PoP and DAGA interaction | Implementing DAGA | Improving DAGA | Conclusion
Proof problem

• **Anonymity through a client OR proof:**
  → I know (private key 1 OR private key 2 OR ... )

• **Growth** $O(6\times n)$, $n = \#members$
  → 32768 members / 32 servers
  • Proof ~6.3 MB, total cost ~200 MB → ~20% of total
Improving the proof

- Work with Kasra Edalatnejadkhamene, PhD student
- Survey of the field
- Split the proof
  - Proof of membership: Accumulator
  - Proof of knowledge: Signature of knowledge
- No concrete scheme
Overview

• Introduction

• PoP and DAGA interaction

• Implementing DAGA

• Improving DAGA

• Conclusion & Future work
Conclusion & Future work

- DAGA and PoP can work together
- Complete Go implementation of DAGA
- Improvement guidelines for the proof
- Next steps
  - Integrate DAGA and PoP
  - Optimize network consumption
  - Continue the work on the proof
  - Improve implementation resistance (secure memory management, constant-time, … )
Distributed randomness

C_s + Signatures

C_s + Signatures

C_0 + Signature

C_i + Signature

\ldots

C_m + Signature

Signature checks

c_i

c_0

\ldots

c_m

Opening checks

C_s + Signature + (C_0, \ldots, C_m) + (c_0, \ldots, c_m)

Computation checks

\ldots

C_s + Signatures + (C_0, \ldots, C_m) + (c_0, \ldots, c_m)

Request challenge
Context

- User public keys (#members)
- Server public keys (#servers)
- Server random commitments (#servers)
- Client random generators (#members)
Accumulator

- **Accumulators from Bilinear Pairings and Applications**
  L. Nguyen, 2005

- **Adjustments:**
  - Trusted setup
  - Bounded
  - Efficiency based on trusted authority
Ring signature

\[ y_r = g_r \left( x_r \right) \]

\[ z = v \]

\[ y_1 = g_1 \left( x_1 \right) \]

\[ y_2 = g_2 \left( x_2 \right) \]

\[ y_3 = g_3 \left( x_3 \right) \]

- *How to Leak a Secret*, R. Rivest, A. Shamir and Y. Tauman