Integrating DAGA into the cothority framework and using it to build a login service

DEDIS, EPFL 2018/19 - Lucas Pires
Responsible: Prof. Bryan Ford, Dr. Ewa Syta
Supervisor: Linus Gasser
Integrating DAGA into the cothority framework and using it to build a login service

**Deniable Anonymous Group Authentication**

- Decentralized Authentication Protocol
- Forward-security, etc. more later
Motivation / Intro

• Authentication Identification and Privacy
• ➔ where possible, get rid of identification
• ➔ DAGMA
• GOAL: offer easy way to use DAGMA, Login Service
Overview

• Background / DAGA
• Couthority implementation
• Authentication delegation
• PoC & demo
• Conclusion
Background / DAGA

Big picture - Properties - Description
Background / DAGA –
Background / DAGA –

Entity / user

DAGA

Anytrust servers
Background / DAGA –

Big picture
Properties
Description

Group

DAGA

Entity / user

Auth. request
Decision

Anytrust servers
Background / DAGA –

- Completeness
- Soundness

DAGA

Entity / user  
Auth. request  
Decision  
Anytrust servers

Group
Background / DAGA –

- Completeness
- Soundness
- Anonymity

DAGA

Entity / user

Group

Auth. request

Decision

Anytrust servers
Background / DAGA –

• Anonymity
• Proportionality

Entity / user

DAGA

Auth. request
Decision
+ Linkage Tag

Anytrust servers

Big picture
Properties
Description
Background / DAGA –

- Anonymity
- Proportionality
- Deniability

Entity / user → Auth. request → Decision → + Linkage Tag → Anytrust servers

DAGA

Group
Background / DAGA –

• Anonymity
• Proportionality
• Deniability
• Forward security

Entity / user → DAGA

Decision + Linkage Tag

Auth. request

Anytrust servers

Group
Background / DAGA –

Big picture

Properties

Description

Prover

Build request / client’s protocol

Context

Verifiers

Adapted / redrawn from https://github.com/dedis/student_17/blob/master/pfs_pop/presentation_pfs_pop.pdf
Background / DAGA –

Big picture

Properties

Description

Build request / client’s protocol

Initial tag

Prover

Context

Verifiers

Adapted / redrawn from https://github.com/dedis/student_17/blob/master/pfs_pop/presentation_pfs_pop.pdf
Background / DAGA –

Build request / client’s protocol

Initial tag → Proof generation

Σ Challenge

Σ commitments

Distributed randomness / challenge generation

Adapted / redrawn from https://github.com/dedis/student_17/blob/master/pfs_pop/presentation_pfs_pop.pdf
Background / DAGA –

Build request / client’s protocol

Initial tag \( \sum \) Challenge \( \sum \) commitments

Proof generation

\( \sum \) Challenge

Request (with \( \sum \) responses)

Distributed randomness / challenge generation

Servers’ protocol

Linkage tag

Collective proof verification, decision and Tag building

Adapted / redrawn from https://github.com/dedis/student_17/blob/master/pfs_pop/presentation_pfs_pop.pdf
Overview

- Background / DAGA
- Cothority implementation
- Authentication delegation
- PoC demo
- Conclusion & Future
Cothority Implementation

• DAGA Library (continuation of A. Villard’s work)
• New Service & Protocols
  (context generation / challenge generation / DAGA servers’ protocol)
• Can run simulations locally and on DETERLab
• 80% code coverage
• Possible to generate proto files
• CLI client
Cothority Implementation

- DAGA Library (continuation of A. Villard’s work)
- **New Service & Protocols**
  (context generation / challenge generation / DAGA servers’ protocol)
- Can run simulations locally and on DETERLab
- 80% code coverage
- Possible to generate proto files
- CLI client
Client / 3rd party service admin

**Administrative phase**

1) Collect public **keys** of subscribers
2) Build a **roster** of willing conodes (partnerships or open access nodes)
Client / 3rd party service admin

1) Collect public keys of subscribers
2) Build a roster of willing conodes (partnerships or open access nodes)
3) Call CreateContext(keys, roster)

Context generation protocol

Random node

Other nodes
Client / 3rd party service admin

Administrative phase

1) Collect public keys of subscribers
2) Build a roster of willing conodes (partnerships or open access nodes)
3) Call CreateContext(keys, roster)

Context generation protocol

New Cothority
For the new context
Build auth. Message $M$

Initial tag

Proof generation

Call $\text{PKClient}(\sum \text{commitments}, \text{Entity})$

$\sum \text{Challenge}$

Call $\text{Auth}(M, \text{Entity})$

$\text{Server’s protocol}$

Need to keep state across endpoint calls

$\rightarrow$ avoid by storing it in clients
Cothority Implementation

• DAGA Library (continuation of A. Villard’s work)
• New Service & Protocols
  (context generation / challenge generation / DAGA servers’ protocol)
• Can run simulations locally and on DETERLab
• 80% code coverage
• Possible to generate proto files
• CLI client
Simulation results – total authentication time

1) DETERLab Setup:
   - pc2133 nodes:
     - Ubuntu 14.04, AMD64
     - CPU: 4 @ 2.13 GHz
     - RAM: 4 GiB
     - LAN with 100 ms delay

2) Local Setup:
   - Debian 9, AMD64
   - CPU: 8 @ 2.50GHz
   - RAM: 16 GiB
Original results and previous student’s results

Original paper (2014)

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

Previous student

Setup:
- Windows 10
- x86-64
- 1 thread @4,5GHz

Taken from https://github.com/dedis/student_17/blob/master/pfs_pop/presentation_pfs_pop.pdf
Simulation results – total authentication time

Local 4 servers

Local 16 servers

Wall time [s]

Number of group members

Number of group members
Simulation results – total server traffic

Previous student’s results
Cothority Implementation

- DAGA Library (continuation of A. Villard’s work)
- New Service & Protocols
  (context generation / challenge generation / DAGA servers’ protocol)
- Can run simulations locally and on DETERLab
- 80% code coverage
- Possible to generate proto files
- CLI client
Overview

• Background / DAGA
• Cothority implementation
• Authentication delegation
• PoC demo
• Conclusion & Future
Authentication delegation

Entity / user

DAGA couthority
Authentication delegation

Entity / user

Service Provider

DAGA cothority
Authentication delegation

Entity / user

Service Provider

DAGA cothority
Authentication delegation
OpenID connect authentication – “code flow”
GET rp/login

OpenID connect authentication
- “code flow”
OpenID connect authentication

“code flow”
OpenID connect authentication - “code flow”
OpenID connect authentication - “code flow”
GET rp/login
REDIRECT IdP/daga_auth
GET IdP/daga_auth
200 authentication page
GET rp/login

REDIRECT IdP/daga_auth

GET IdP/daga_auth

200 authentication page
GET rp/login

REDIRECT IdP/daga_auth

DAGA client daemon

Browser / WEB UI

GET IdP/daga_auth

200 authentication page
Call \textit{PKClient}(commitments)

Browser / WEB UI

Arguments, context + key

DAGA client daemon

IdP

Challenge
Call **PKClient**(commitments)

**DAGA client daemon**

**Auth. Msg**

**Browser / WEB UI**

Arguments, context + key

**Challenge**
DAGA client daemon

Call **PKClient**(commitments)

Arguments, context + key

Auth. Msg

Browser / WEB UI

POST back with Auth. msg

Challenge
Call **PKClient**(commitments)

Arguments, context + key

POST back with Auth. msg

Call **Auth**(Auth. msg)

Challenge

Linkage Tag
Call **PKClient**(commitments)

POST back with Auth. msg

GET rp/callback with **code**

REDIRECT rp/callback with **code**

Call **Auth**(Auth. msg)

Linkage Tag

Challenge
Call **PKClient**(commitments)

GET `rp/callback` with **code**

POST `IdP/token_endpoint` with **code**

REDIRECT `rp/callback` with **code**

**200 token**

POST back with Auth. msg

Call **Auth**(Auth. msg)

GET `rp/callback` with **code**

Challenge

Linkage Tag
Demo
Conclusion

• Democratization of DAGA as anonymous authentication is feasible
• Future works:
Conclusion

• Democratization of DAGA as anonymous authentication is feasible
• Future works:
  • Need ways to manage partnerships and evolve contexts
Conclusion

• Democratization of DAGA as anonymous authentication is feasible

• Future works:
  • Need ways to manage partnerships and evolve contexts
  • Need ways to scale (random sub-groups)
Conclusion

- Democratization of DAGA as anonymous authentication is feasible

- Future works:
  - Need ways to manage partnerships and evolve contexts
  - Need ways to scale (random sub-groups)
  - Need to armor everything (memory protection, ...)

1) GET /signin

2) HTTP redirect IdP/daga_auth

3) Input user key and auth. context

4) Go ! => transmit order to local daemon (websocket)

5) Context + key

6) PKClient commitments

7) PKClient challenge

8) auth. message

9) POST back auth. message

10) Auth(auth. message)

11) Tag

12) HTTP redirect RP with authorization code

13/14) receive ID token in exchange of authorization code

15) logs user in

User-Agent
Browser
DAGA
DEX IdP
Service Provider / RP

/local daga daemon

/missing.png

Deniable Anonymous Group Authentication

Choose daga auth. context:
context : Browse... No file selected.
client : Browse... No file selected.

Go !