Adding Secure Transparency Logging to the Prime Core

Hans Hedbom, Tobias Pulls, Peter Hjärtquist and Andreas Lavén

Karlstad University

Background

- Prime Life
  - Multidisciplinary Consortium of 13+ organisations
  - Vision: **counter the trend to life-long personal data trails without compromising on functionality**.
  - Task 2.2.1 about transparency tools for privacy

Why Transparency tools?

- New technologies makes it hard to controll access and the existence of data
  - AmI, Data Mining, Web 2.0, Online Comunities
  - Control use as a complement to Consealment?
- European Law requires transparency
  - Not online requirement, but online tools will probably make things more costeffective.

Why privacy preserving secure logs?

- A need to know how data been handled.
- Realtime access for data subjects to processing and access history.
- Detection of policy violations.
- Should not reveal new personal information to others.

Related work.

- Schneier and Kelsey: Discusses secure logs using hash chains and describes an algorithm and a protocol.
- Holt: Improvements to S-K log making it publicly integrity verifiable. Discusses different uses of public key cryptography.
- Ma and Tsudik: Solutions to integrity problems in S-K logs using forward secure sequential aggregation
- Accorsi et. al: Privacy Policy violation detection using S-K logs. username part of the key.
- Missing: Unlinkability, Secure anonymous access, Inability for server to read entries after commit.

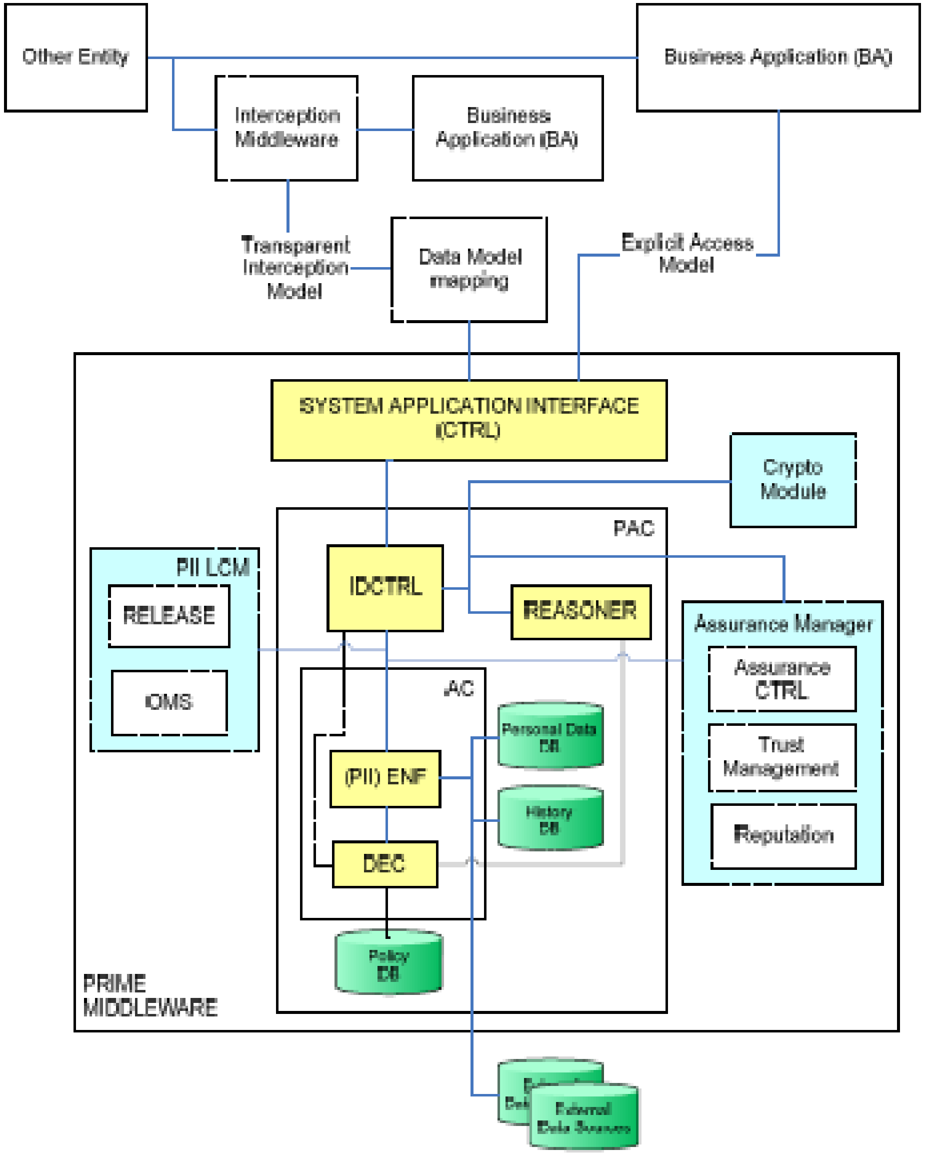
Assumptions

- Assumptions
  - Logging environment (Prime Core) plays by the rules (but can turn bad).
  - Nothing can be done for future log entries if an attacker has full control of the environment or if the server turns bad (forward security).

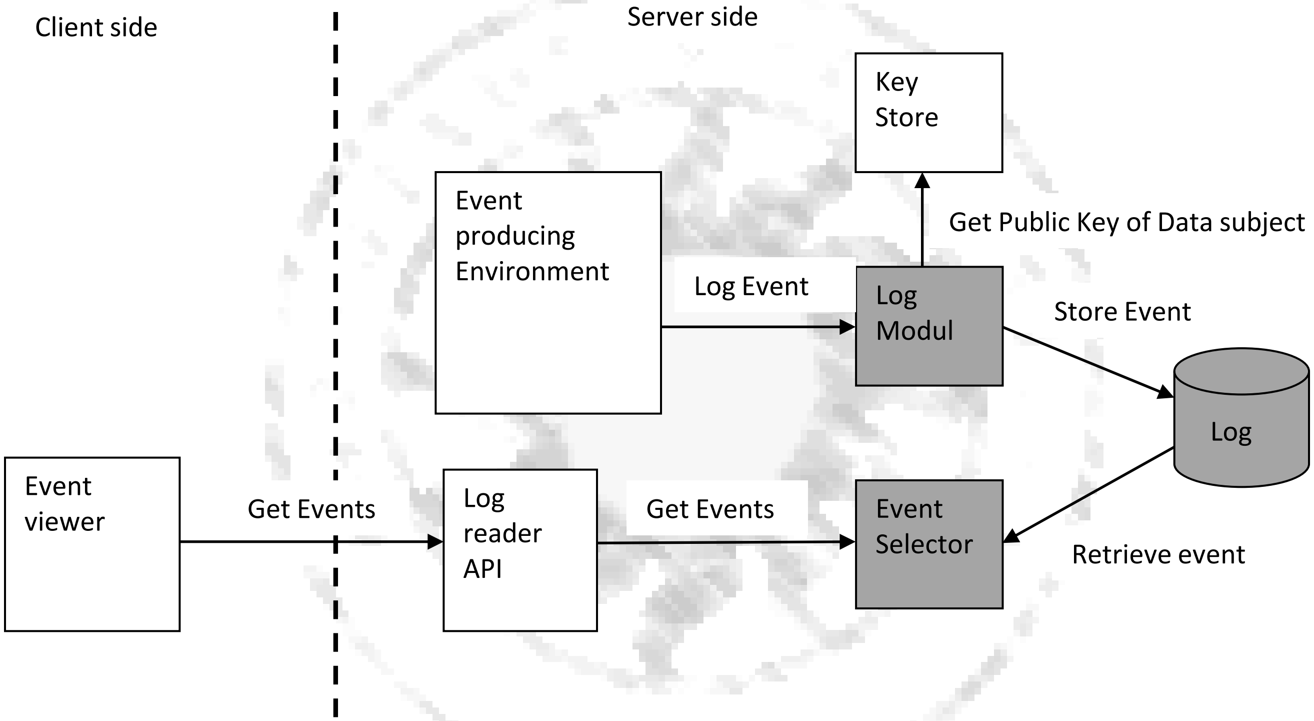
Requirements

- • It should not be possible for anybody except the data subject to decrypt log entries once they are committed to the log.
- • It should not be possible to alter nor remove entries made prior to an attacker taking control of the data controller without detection.
- • It should not be possible to link more than one log entry in the log referring to a specific data subject with that data subject except by the data subject itself.
- • For efficiency reasons the solution should as far as possible not require that the whole log database is fully traversed by any entity or sent as a whole to the data subject.

The Prime Core



Log Architecture



Secrets

- Secrets known and stored by the server:
  - SAS$_0$- A random number constituting the initial server secret used to authenticate all entries in the log for the server.
  - ServerID$_0$- A random number constituting the initial ServerID seed.
- Secret known and stored by each client for each data subject identifier used by a data subject using the client:
  - DSS$_0$- A random number constituting the initial client secret used to authenticate all entries relating to the data subject identifier for the client.
  - EntryID$_0$- A random number constituting the client's initial EntryID seed for the data subject identifier.

State Tables

Every update overwrites and irretrieveably deletes the previous value

- The server state table:
  - **SAS_{j+1}** the secret used for the next entry
  - **Latest ServerChain** to build the next ServerChain
  - **Latest ServerID** to build the next ServerID

- The (server's) data subject state table:
  - **Data Subject Identifier**
  - **DSS_{i+1}** the secret used for the next entry
  - **Latest DataSubjectChain** to build the next ClientChain
  - **Latest EntryID** *for that client* to build the next EntryID

Adding an event to the Log

DataSubjectChain$_{i-1}$

EntryID$_{i-1}$

DSS$_i$

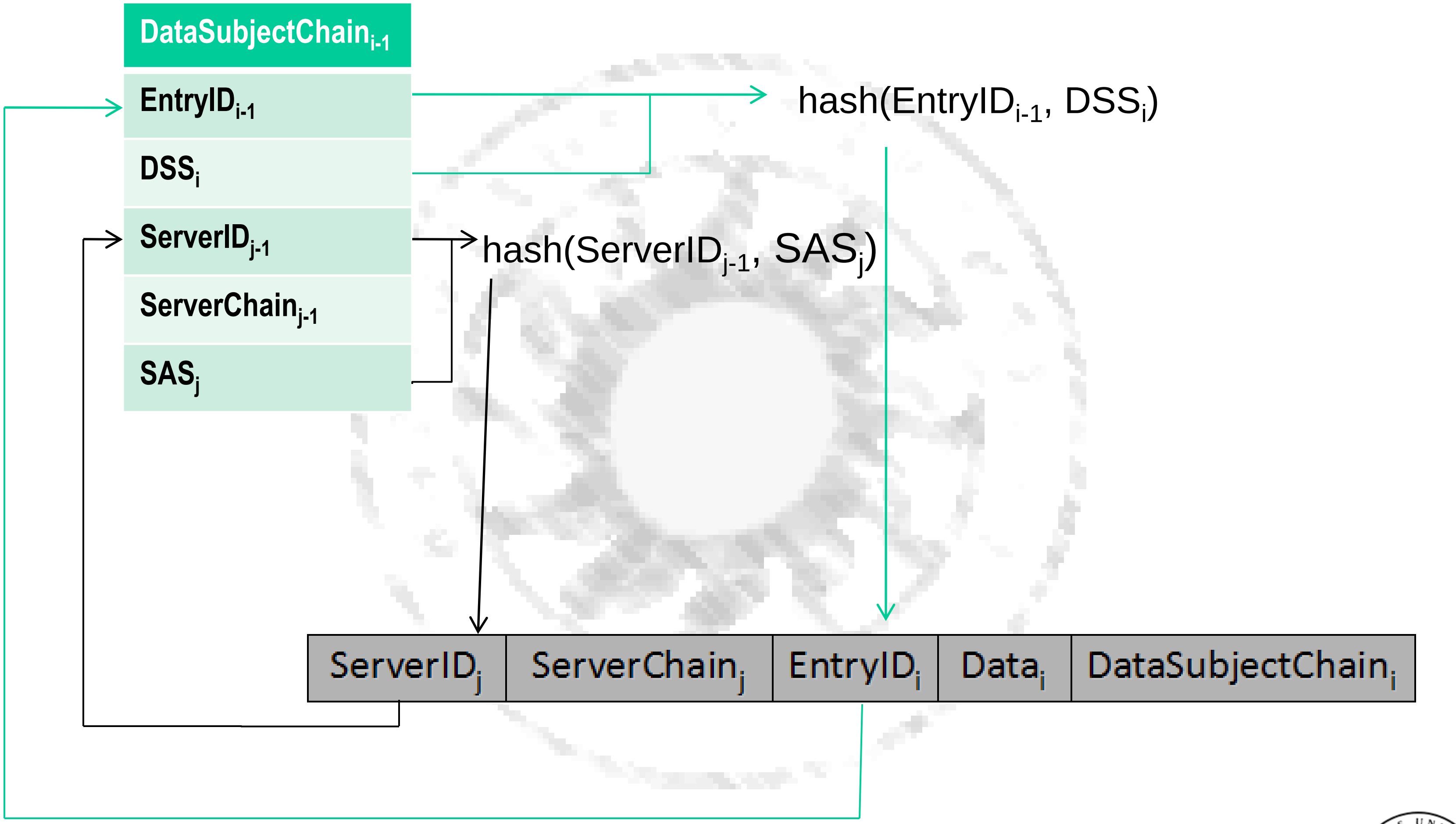
ServerID$_{j-1}$

ServerChain$_{j-1}$

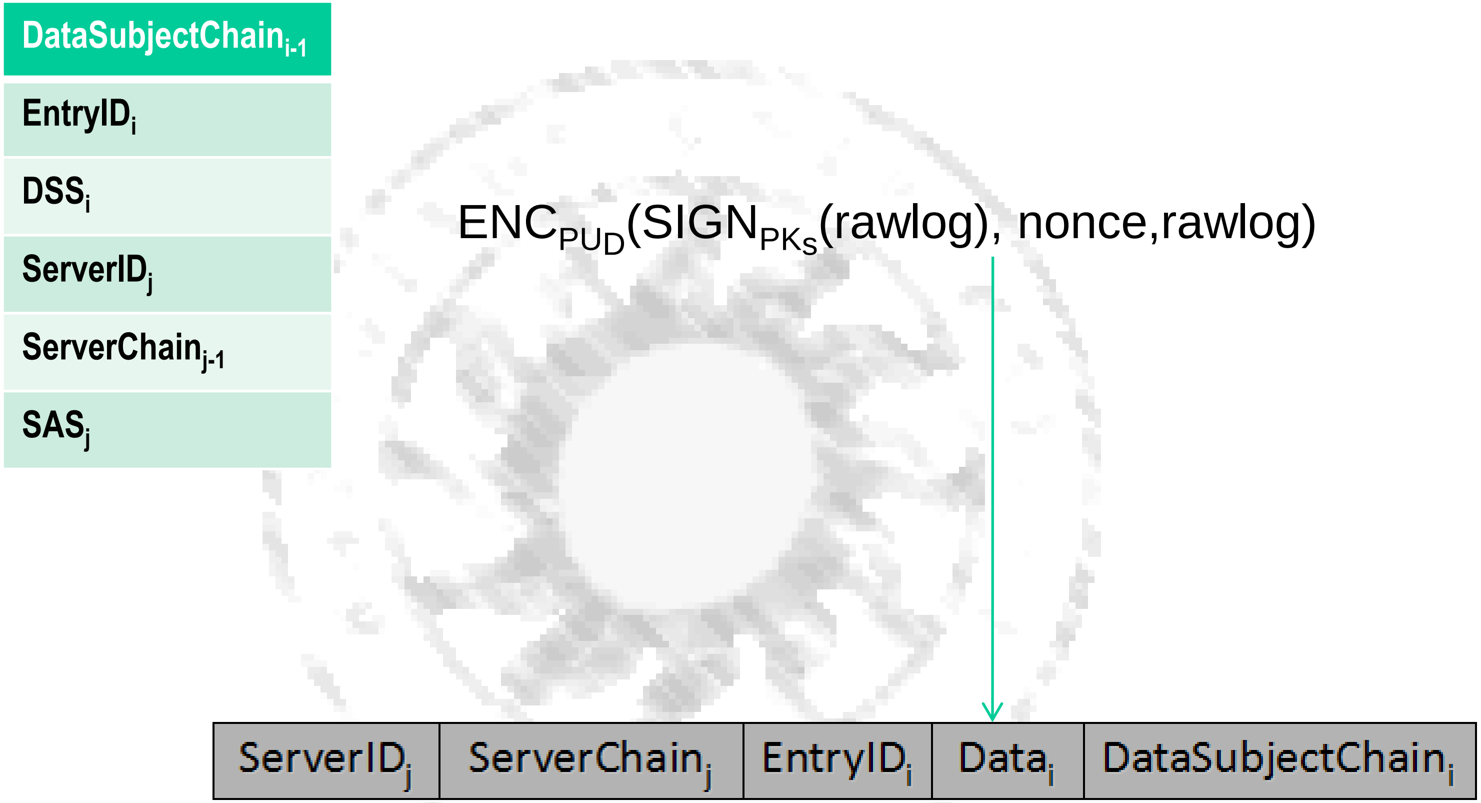
SAS$_j$

ServerID$_j$	ServerChain$_j$	EntryID$_i$	Data$_i$	DataSubjectChain$_i$

Adding an event to the Log



Adding an event to the Log



DataSubjectChain$_{i-1}$

EntryID$_i$

DSS$_i$

ServerID$_j$

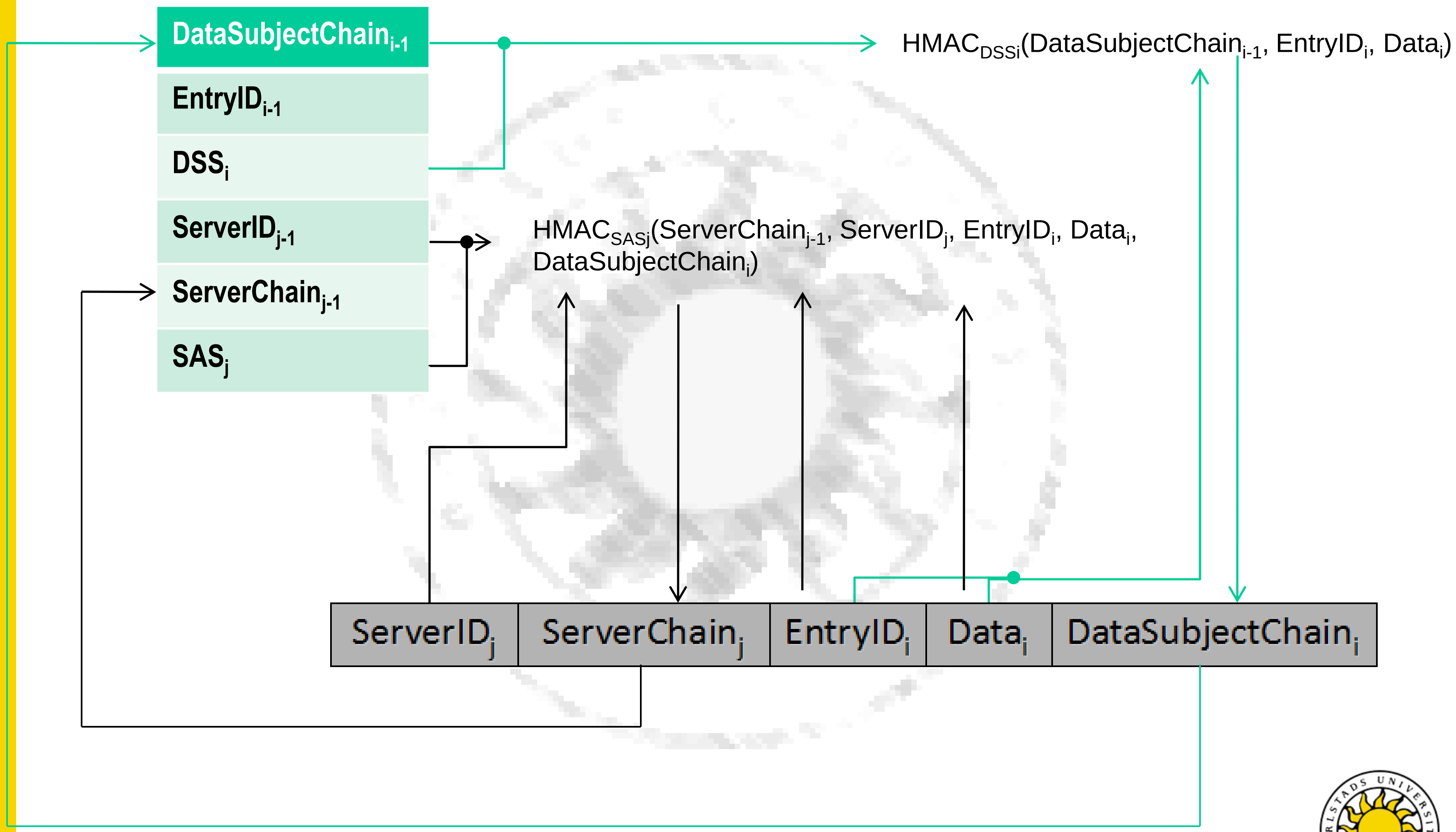
ServerChain$_{j-1}$

SAS$_j$

$\text{ENC}_{\text{PU}_\text{D}}(\text{SIGN}_{\text{PK}_\text{S}}(\text{rawlog}), \text{nonce}, \text{rawlog})$

ServerID$_j$	ServerChain$_j$	EntryID$_i$	Data$_i$	DataSubjectChain$_i$

Adding an event to the Log



$\text{DataSubjectChain}_{i-1}$

EntryID_{i-1}

DSS_i

ServerID_{j-1}

ServerChain_{j-1}

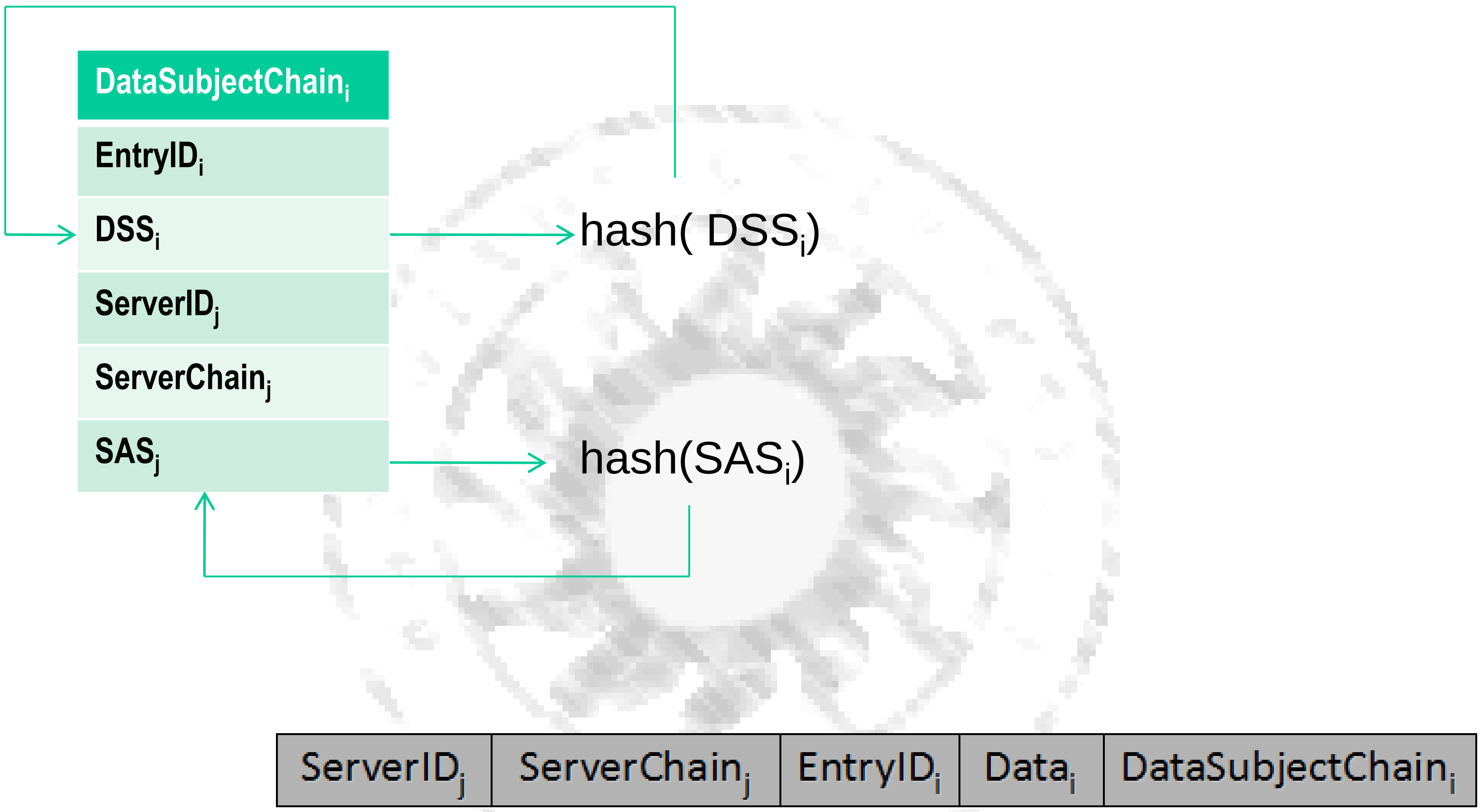
SAS_j

$\text{HMAC}_{\text{DSSi}}(\text{DataSubjectChain}_{i-1}, \text{EntryID}_i, \text{Data}_i)$

$\text{HMAC}_{\text{SASj}}(\text{ServerChain}_{j-1}, \text{ServerID}_j, \text{EntryID}_i, \text{Data}_i,$
$\text{DataSubjectChain}_i)$

ServerID_j | ServerChain_j | EntryID_i | Data_i | $\text{DataSubjectChain}_i$

Adding an event to the Log



$DataSubjectChain_i$

$EntryID_i$

DSS_i

$ServerID_j$

$ServerChain_j$

SAS_j

hash(DSS_i)

hash(SAS_i)

$ServerID_j$	$ServerChain_j$	$EntryID_i$	$Data_i$	$DataSubjectChain_i$

Adding an event to the Log

$\text{DataSubjectChain}_i$

EntryID_i

DSS_{i+1}

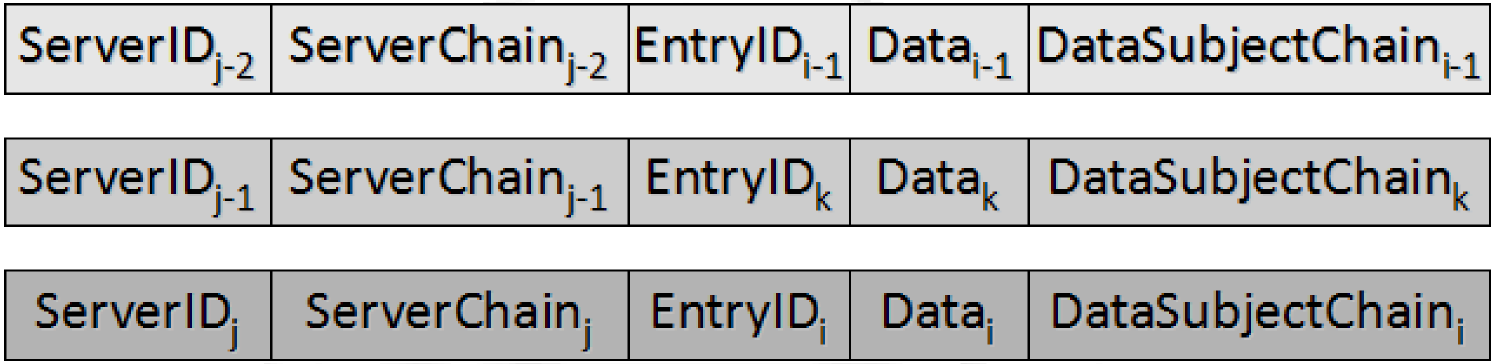
ServerID_j

ServerChain_j

SAS_{j+1}

ServerID_j	ServerChain_j	EntryID_i	Data_i	$\text{DataSubjectChain}_i$

Structure



ServerID$_{j-2}$	ServerChain$_{j-2}$	EntryID$_{i-1}$	Data$_{i-1}$	DataSubjectChain$_{i-1}$
ServerID$_{j-1}$	ServerChain$_{j-1}$	EntryID$_{k}$	Data$_{k}$	DataSubjectChain$_{k}$
ServerID$_{j}$	ServerChain$_{j}$	EntryID$_{i}$	Data$_{i}$	DataSubjectChain$_{i}$

The Log Reader Api

- 1. GetLogEntry(EntryID) - returns the object(s) with the supplied EntryID. Since only a data subject knowing the right private key can decrypt the data field this method does not need the data subject to be identified.

- 2. GetLatestEntryID(DataSubjectIdentifier) - returns a data structure containing the EntryID in the data subject state table for the data subject identifier and a nonce. The structure is encrypted with the public key stored for the data subject. Returns dummy responces for invalid Data Subject Identifiers.

Validation

- Both the server and the client are able to validate the log, without each other’s secrets

Validation, server

- Calculate chain
  - $\mathrm{HMAC}_{\mathrm{ServerAccessSecret}}($ previous calculated chain,
    client chain,
    encrypted data in log entry,
    entry identifier,
    server identifier)
- Compare the calculated chain with the chain stored in the log entry

Validation, client

- Calculate chain
  - $\text{HMAC}_{\text{DataSubjectSecret}}$ (previous client chain,
    identifier of the entry,
    encrypted data in log entry)
- Compare the calculated chain with the chain stored in the log entry
- Check that the decrypted data is signed by the server

Attack scenarios

- Items of interest:
  - The log table
  - Server state which contains the server state table, the data subject state table and the private key used to sign entries
  - SAS_0 and $ServerID_0$
  - DSS_0 and $DataSubjectEntry_0$ for every client
  - The private key for each client

Issues

- Client behavior important
  - Access pattern
  - Anonymous network
- Authentication keys reused
  - ID and Chain entries use the same key i.e DSS or SAS. -> Problems if Hash broken.

Conclusion and Future work

- We have presented a privacy friendly secure log
- The log builds on previous work
- Addresses problems of linkabillity, anonymous access and reversibility once committed.
- The log is implemented.
- Next step to integrate into PRIME Core, to implement and design log Viewer and to estimate optimal log pattern.
- ”Log data propagation”

Conclusion and Future work

- Open questions for future work
  - 1. Is it really possible to irreversibly overwrite the old authentication keys once stored in the server's state i.e. memory?
  - 2. Will the actual database used to store the log entries to some extent leak the order in which the entries were added to the database due to some internal structures or functionality?

Questions

?