Modelling Secure Multi User Deduplication - S2Dedup
Work Assignment
CSI

Bruno Gião	Tiago Teixeira	João Silva
PG57858	PG57904	PG55618

June 24, 2025

Contents

1. Introduction
   - Context
2. Secure Deduplication
   - Relational Logic
3. Alloy
   - Verifying Actions
4. Model Validation
5. Conclusion
6. Bibliography

Deduplication

- **Deduplication** is a known storage systems optimization that reduces used space to great effect.
- Deduplication consists of finding identical **chunks**, which might be blocks or files, and mapping a logical address to a physical address and a hash of the chunk's content with its physical address to a number of references. Paulo and Pereira, 2014
- However, its usage in multi-user systems[1] raises security concerns due to the "merging" and removal of cross-client data.

[1] See **Dropbox** and other **Amazon S3** solutions

Deduplication

- **Deduplication** is a known storage systems optimization that reduces used space to great effect.
- Deduplication consists of finding identical **chunks**, which might be blocks or files, and mapping a logical address to a physical address and a hash of the chunk's content with its physical address to a number of references. Paulo and Pereira, 2014
- However, its usage in multi-user systems[1] raises security concerns due to the "merging" and removal of cross-client data.

[1] See **Dropbox** and other **Amazon S3** solutions

Deduplication

- **Deduplication** is a known storage systems optimization that reduces used space to great effect.
- Deduplication consists of finding identical **chunks**, which might be blocks or files, and mapping a logical address to a physical address and a hash of the chunk's content with its physical address to a number of references. Paulo and Pereira, 2014
- However, its usage in multi-user systems[1] raises security concerns due to the "merging" and removal of cross-client data.

[1] See **Dropbox** and other **Amazon S3** solutions

Encryption

- A possible solution to the issue presented in the previous slide is to apply encryption to user data.
- However, some methods of encryption might reduce deduplication efficiency, or make the system less secure by producing deterministic ciphertexts.
- It is also worth noting that some disk encryption modes are not usually preferred in storage systems, having a focus towards using length preserving modes such as XTS or GSM. As such, we consider encrypted blocks to be length-preserved cyphertexts of original blocks. Stallings, 2010

Encryption

- A possible solution to the issue presented in the previous slide is to apply encryption to user data.
- However, some methods of encryption might reduce deduplication efficiency, or make the system less secure by producing deterministic ciphertexts.
- It is also worth noting that some disk encryption modes are not usually preferred in storage systems, having a focus towards using length preserving modes such as XTS or GSM. As such, we consider encrypted blocks to be length-preserved cyphertexts of original blocks. Stallings, 2010

Encryption

- A possible solution to the issue presented in the previous slide is to apply encryption to user data.
- However, some methods of encryption might reduce deduplication efficiency, or make the system less secure by producing deterministic ciphertexts.
- It is also worth noting that some disk encryption modes are not usually preferred in storage systems, having a focus towards using length preserving modes such as XTS or GSM. As such, we consider encrypted blocks to be length-preserved cyphertexts of original blocks. Stallings, 2010

S2Dedup

- S2Dedup is a solution that intends to approach this issue using trusted execution environments, namely Intel SGX.
- With this, the system can allow for probabilistic encryption from the clients and deterministic re-encryption, provided decrypting and re-encryption are done at SGX enclaves.

S2Dedup

- S2Dedup is a solution that intends to approach this issue using trusted execution environments, namely Intel SGX.
- With this, the system can allow for probabilistic encryption from the clients and deterministic re-encryption, provided decrypting and re-encryption are done at SGX enclaves.

Proposed Work

- Our intention is, provided S2Dedup's architecture, formally model the system, for which we shall consider starting by modeling an unsecure multi-user deduplication and Secure Deduplication using a Plain security scheme.
- We consider files, unique to clients/users, that are composed of blocks. Our deduplication defines the chunk at the block level.
- As defined in S2Dedup's paper(Miranda et al., 2021), the Plain scheme consists of probabilistic encryption from the client, hash calculation and deterministic re-encryption at the SGX enclaves.

Proposed Work

- Our intention is, provided S2Dedup's architecture, formally model the system, for which we shall consider starting by modeling an unsecure multi-user deduplication and Secure Deduplication using a Plain security scheme.
- We consider files, unique to clients/users, that are composed of blocks. Our deduplication defines the chunk at the block level.
- As defined in S2Dedup's paper(Miranda et al., 2021), the Plain scheme consists of probabilistic encryption from the client, hash calculation and deterministic re-encryption at the SGX enclaves.

Proposed Work

- Our intention is, provided `S2Dedup`'s architecture, formally model the system, for which we shall consider starting by modeling an unsecure multi-user deduplication and Secure Deduplication using a Plain security scheme.
- We consider files, unique to clients/users, that are composed of blocks. Our deduplication defines the chunk at the block level.
- As defined in `S2Dedup`'s paper(Miranda et al., 2021), the Plain scheme consists of probabilistic encryption from the client, hash calculation and deterministic re-encryption at the `SGX` enclaves.

Secure Deduplication

Relational Diagram

$$
\begin{tikzcd}
 & File \arrow[r, "Blocks"'] \arrow[d, dashed, shift right, "active"'] \arrow[d, dashed, shift left, "written"] & Block \arrow[dll, bend right, "hash"'] & \\
Hash & Bool & User \arrow[ul, dashed, "Files"] & EncBlock \times User \arrow[l, "\pi_2"] \arrow[ul, "Decrypt"'] \arrow[d, "\pi_1"'] \\
Hash \times \mathbb{Z}^- \arrow[u, "\pi_1"] \arrow[r, "\pi_2"] \arrow[d, "Index"] & \mathbb{Z}^- \arrow[u, "freeBlock"] & User \times Block \arrow[u, "\pi_1"] \arrow[uu, bend left, "\pi_2"] \arrow[r, "userEnc"] & EncBlock \\
\mathbb{N} & & &
\end{tikzcd}
$$

Figure: Relational Diagram for Secure Deduplication

MAGIC SQUARE

$$\begin{tikzcd}[column sep=huge, row sep=huge]
\mathsf{User} \times \mathsf{Block} \arrow[d, "\pi_2"'] & \mathsf{EncBlock} \times \mathsf{User} \arrow[l, "(\mathit{userEnc}^\circ \cdot \pi_1) \cap \langle \pi_2, \top \rangle"'] \arrow[d, "\mathit{id}"] \arrow[dl, phantom, "\subseteq"] \\
\mathsf{Block} & \mathsf{EncBlock} \times \mathsf{User} \arrow[l, "\mathit{Decrypt}"]
\end{tikzcd}$$

FIGURE: Magic Square for decryption and encryption

Logical Properties

$$\textbf{Files Entire: } id \subseteq Files^{\circ} \cdot Files \tag{1}$$

$$\textbf{Files Injective: } Files^{\circ} \cdot Files \subseteq id \tag{2}$$

$$\textbf{Files Surjective: } id \subseteq Files \cdot Files^{\circ} \tag{3}$$

$$\textbf{Blocks Entire: } id \subseteq Blocks^{\circ} \cdot Blocks \tag{4}$$

$$\textbf{Blocks Surjective: } id \subseteq Blocks \cdot Blocks^{\circ} \tag{5}$$

Logical Properties

$$\textbf{Index Simple:}\ \ Index \cdot Index^\circ \subseteq id \tag{6}$$

$$\textbf{userEnc Entire:}\ \ id \subseteq userEnc^\circ \cdot userEnc \tag{7}$$

$$\textbf{userEnc Simple:}\ \ userEnc \cdot userEnc^\circ \subseteq id \tag{8}$$

$$\textbf{Decrypt Simple:}\ \ Decrypt \cdot Decrypt^\circ \subseteq id \tag{9}$$

$$\textbf{Decrypt Surjective:}\ \ id \subseteq Decrypt \cdot Decrypt^\circ \tag{10}$$

The Hash Function

- Since a Hash function is naturally a computational concept, any practical implementation will always have limitations.
- **On injectivity:** One notable limitation of hash functions is the presence of **collisions**.
- As such, in a real world scenario, hash functions are not injective, despite it being an idealized property; common practice is to **reduce** the probability of collisions occurring.
- Since this is a mathematical formulation, we can and will consider an ideal perfect hash function where no collisions are possible.

The Hash Function

$$\textbf{hash Simple:}\ \ hash \cdot hash^{\circ} \subseteq id \tag{11}$$

$$\textbf{hash Entire:}\ \ id \subseteq hash^{\circ} \cdot hash \tag{12}$$

$$\textbf{hash Injective:}\ \ hash^{\circ} \cdot hash \subseteq id \tag{13}$$

Actions - Storing

1. **Send a File (`send_file`):** File must neither be active nor writen. In which case, activates the file to be sent.
2. **Send File blocks (`send_block`):** The non written, activated file's block's hash cannot be indexed and there must be a free physical address.
3. **Send Deduplicated File blocks (`send_block_dedup`):** Sends a block of a non-written file, indexed in its hash, updating the index reference number.
4. **Set File Send as done (`clean_done`):** When all the blocks are sent, this action sets the file as done.

Actions - Storing

1. **Send a File (`send_file`):** File must neither be active nor writen. In which case, activates the file to be sent.
2. **Send File blocks (`send_block`):** The non written, activated file's block's hash cannot be indexed and there must be a free physical address.
3. **Send Deduplicated File blocks (`send_block_dedup`):** Sends a block of a non-written file, indexed in its hash, updating the index reference number.
4. **Set File Send as done (`clean_done`):** When all the blocks are sent, this action sets the file as done.

Actions - Storing

1. **Send a File (`send_file`):** File must neither be active nor writen. In which case, activates the file to be sent.
2. **Send File blocks (`send_block`):** The non written, activated file's block's hash cannot be indexed and there must be a free physical address.
3. **Send Deduplicated File blocks (`send_block_dedup`):** Sends a block of a non-written file, indexed in its hash, updating the index reference number.
4. **Set File Send as done (`clean_done`):** When all the blocks are sent, this action sets the file as done.

Actions - Storing

1. **Send a File (`send_file`):** File must neither be active nor writen. In which case, activates the file to be sent.
2. **Send File blocks (`send_block`):** The non written, activated file's block's hash cannot be indexed and there must be a free physical address.
3. **Send Deduplicated File blocks (`send_block_dedup`):** Sends a block of a non-written file, indexed in its hash, updating the index reference number.
4. **Set File Send as done (`clean_done`):** When all the blocks are sent, this action sets the file as done.

Actions - Deletions

1. **Delete File (`elim_file`):** Verifies that a certain file is not active and already written. If so, prepares it to be deleted.
2. **Delete Deduplicated File Blocks (`elim_block`):** Deletes a block of the file being deleted. In this case, the reference number associated by the block's hash is greater than 1, so this number is decremented in the index relation.
3. **Delete File Blocks (`elim_block_clean`):** Deletes a block of the file being deleted. In this case, since the reference number associated by the block's hash is equal to 1, we remove the entry from the index relation, also freeing the mapped physical address.
4. **Set File as Deleted (`clean_cleanDone`):** Occurs when all the blocks of the file are deleted, deactivating the file as a result.

Actions - Deletions

1. **Delete File (`elim_file`):** Verifies that a certain file is not active and already written. If so, prepares it to be deleted.
2. **Delete Deduplicated File Blocks (`elim_block`):** Deletes a block of the file being deleted. In this case, the reference number associated by the block's hash is greater than 1, so this number is decremented in the index relation.
3. **Delete File Blocks (`elim_block_clean`):** Deletes a block of the file being deleted. In this case, since the reference number associated by the block's hash is equal to 1, we remove the entry from the index relation, also freeing the mapped physical address.
4. **Set File as Deleted (`clean_cleanDone`):** Occurs when all the blocks of the file are deleted, deactivating the file as a result.

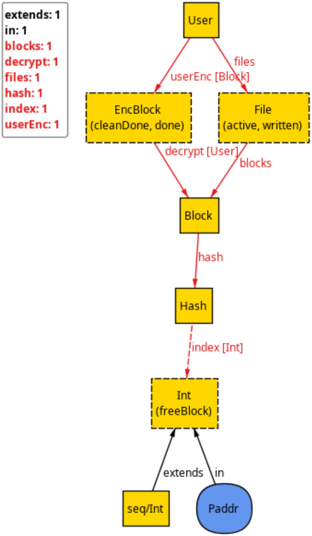
Actions - Deletions

1. **Delete File (`elim_file`):** Verifies that a certain file is not active and already written. If so, prepares it to be deleted.
2. **Delete Deduplicated File Blocks (`elim_block`):** Deletes a block of the file being deleted. In this case, the reference number associated by the block's hash is greater than 1, so this number is decremented in the index relation.
3. **Delete File Blocks (`elim_block_clean`):** Deletes a block of the file being deleted. In this case, since the reference number associated by the block's hash is equal to 1, we remove the entry from the index relation, also freeing the mapped physical address.
4. **Set File as Deleted (`clean_cleanDone`):** Occurs when all the blocks of the file are deleted, deactivating the file as a result.

Actions - Deletions

1. **Delete File (`elim_file`):** Verifies that a certain file is not active and already written. If so, prepares it to be deleted.
2. **Delete Deduplicated File Blocks (`elim_block`):** Deletes a block of the file being deleted. In this case, the reference number associated by the block's hash is greater than 1, so this number is decremented in the index relation.
3. **Delete File Blocks (`elim_block_clean`):** Deletes a block of the file being deleted. In this case, since the reference number associated by the block's hash is equal to 1, we remove the entry from the index relation, also freeing the mapped physical address.
4. **Set File as Deleted (`clean_cleanDone`):** Occurs when all the blocks of the file are deleted, deactivating the file as a result.

Metamodel

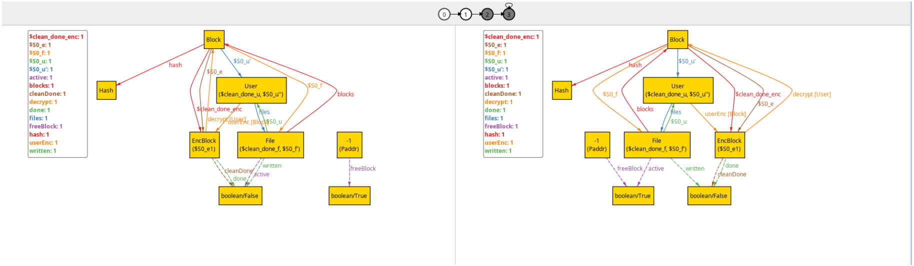

extends: 1
in: 1
blocks: 1
decrypt: 1
files: 1
hash: 1
index: 1
userEnc: 1
User
userEnc [Block]
files
EncBlock
(cleanDone, done)
File
(active, written)
decrypt [User]
blocks
Block
hash
Hash
index [Int]
Int
(freeBlock)
extends
in
seq/Int
Paddr

Run S0

```
run S0{
  some e1:EncBlock|some u:User| some f:File {
    send_file[u,f];send_block[e1,u,f]; clean_done
    }
} expect 1

```

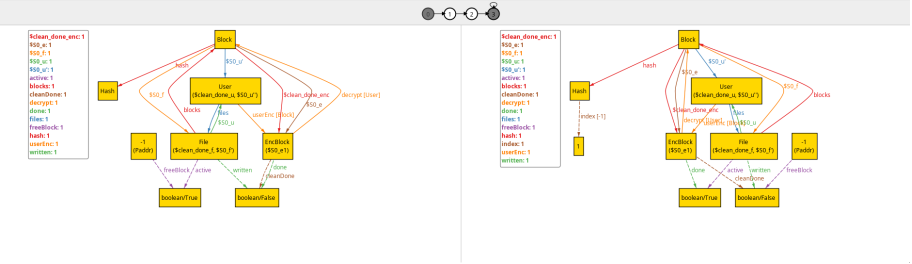
Figure: Run example where an user sends a file with a single original block

Run S0

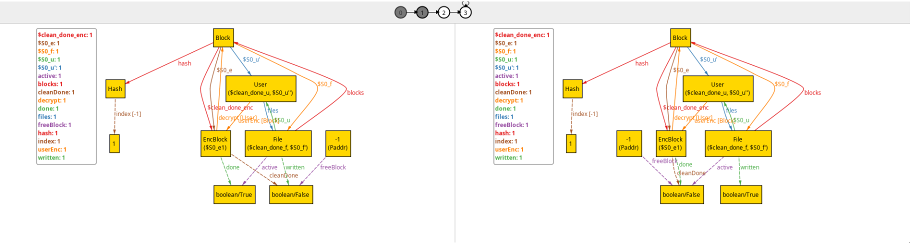


$clean_done_enc: 1
$S0_e: 1
$S0_f: 1
$S0_u: 1
$S0_u': 1
active: 1
blocks: 1
cleanDone: 1
decrypt: 1
done: 1
files: 1
freeBlock: 1
hash: 1
userEnc: 1
written: 1
Block
Hash
User ($clean_done_u, $S0_u'')
EncBlock ($S0_e1)
File ($clean_done_f, $S0_f')
-1 (Paddr)
boolean/False
boolean/True

Run S0


$clean_done_enc: 1
$S0_e: 1
$S0_f: 1
$S0_u: 1
$S0_u': 1
active: 1
blocks: 1
cleanDone: 1
decrypt: 1
done: 1
files: 1
freeBlock: 1
hash: 1
userEnc: 1
written: 1
Block
Hash
User ($clean_done_u, $S0_u'')
-1 (Paddr)
File ($clean_done_f, $S0_f')
EncBlock ($S0_e1)
boolean/True
boolean/False
$clean_done_enc: 1
$S0_e: 1
$S0_f: 1
$S0_u: 1
$S0_u': 1
active: 1
blocks: 1
cleanDone: 1
decrypt: 1
done: 1
files: 1
freeBlock: 1
hash: 1
index: 1
userEnc: 1
written: 1
index [-1]
1

Run S0

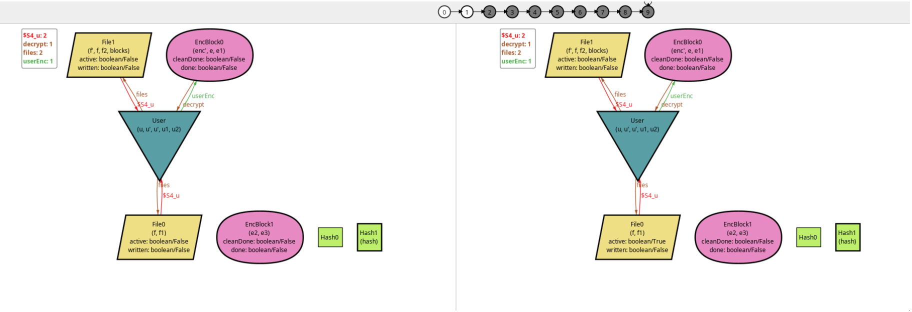

$clean_done_enc: 1
$S0_e: 1
$S0_f: 1
$S0_u: 1
$S0_u': 1
active: 1
blocks: 1
cleanDone: 1
decrypt: 1
done: 1
files: 1
freeBlock: 1
hash: 1
index: 1
userEnc: 1
written: 1
Block
Hash
1
User
($clean_done_u, $S0_u")
EncBlock
($S0_e1)
File
($clean_done_f, $S0_f')
-1
(Paddr)
boolean/True
boolean/False
hash
index [-1]
$S0_u'
$S0_e
$S0_f
blocks
$clean_done_enc
decrypt [User]
userEnc [Block]
files
$S0_u
done
active
written
cleanDone
freeBlock

Run S4

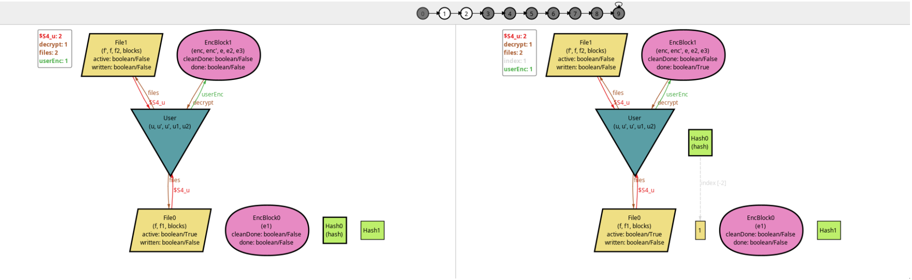
```
run S4 {
  some u2,u1:User | some f1, f2: File |  some e1,e2,e3:
    EncBlock{
     send_file[u2,f1];send_block[e2,u2,f1]; clean_done;
    send_file[u1,f2]; send_block_dedup[e3,u1,f2]; send_block
    [e1,u1,f2]; clean_done; elim_file[u2,f1]; elim_block[e2,
    u2,f1]
        }
} expect 1

```

Figure: Run example where two users send files with identical chunks, where the first user also deletes their file

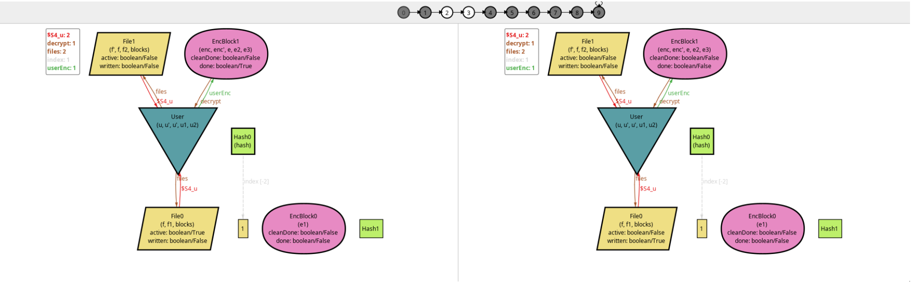
Run S4 Projected over Blocks



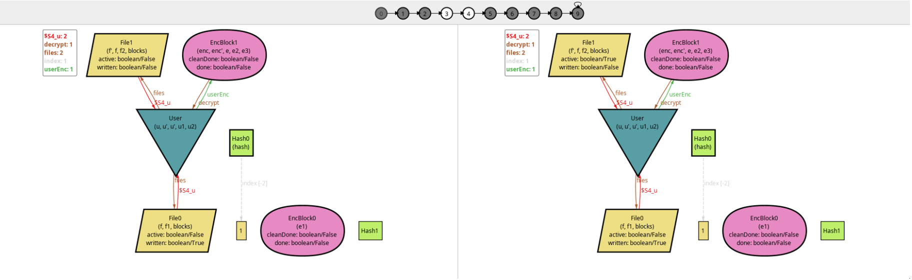
Run S4 Projected over Blocks



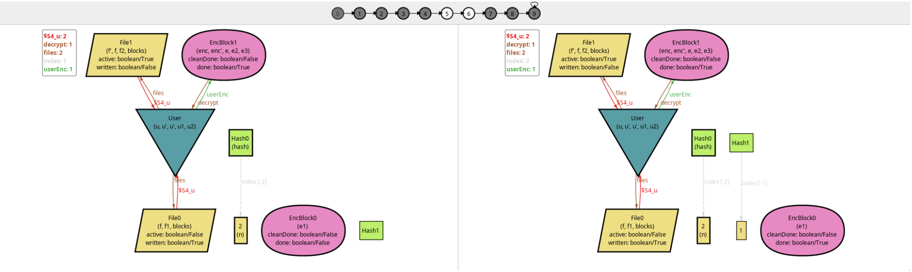
Run S4 Projected over Blocks



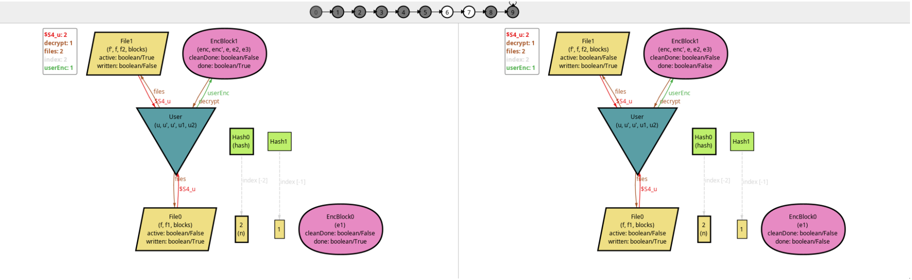
Run S4 Projected over Blocks



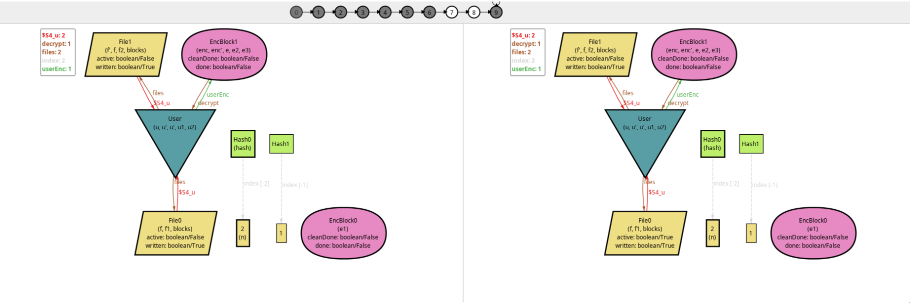
Run S4 Projected over Blocks



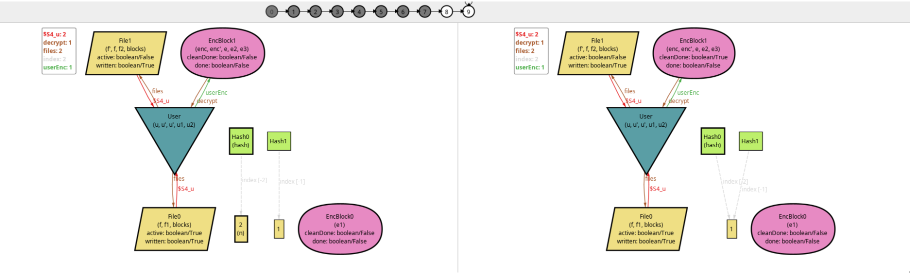
Run S4 Projected over Blocks



Run S4 Projected over Blocks



Run S4 Projected over Blocks



Temporal Properties

```
all f:File | all u:User | always {
    elim_file[u,f]
    implies
    once send_file[u,f]
}
```

Figure: If an User deletes a file, then the User sent that same file previously

```
(always stutter)
or
(some u:User,f:File | stutter until send_file[u,f])
```

Figure: The first action to happen ought to be send_file

```
all f:File | always {
    f->True in written
    implies
    (clean_cleanDone releases f->True in written)
}
```

FIGURE: If a file is written, then it can only be unwritten after the action clean_cleanDone

```
(all u:User | all f:File |
              always {not elim_file[u,f]})
implies
(always True.~written in True.~written')
```

FIGURE: If the deletion of a file never happens, the number of written files never decreases

```
(all u:User | all f:File | always {not send_file[u,f]})
implies
(always no True.~written)

```

FIGURE: If files are never sent, there cannot be written files

Alloy Output

12 commands were executed. The results are:

#1: **Instance found.** S0 is consistent, as expected.
#2: No instance found. S1 may be inconsistent, as expected.
#3: **Instance found.** S2 is consistent, as expected.
#4: **Instance found.** S3 is consistent, as expected.
#5: **Instance found.** S4 is consistent, as expected.
#6: No instance found. Sfail may be inconsistent, as expected.
#7: No instance found. Sfail2enc may be inconsistent, as expected.
#8: No counterexample found. OP1 may be valid.
#9: No counterexample found. OP2 may be valid.
#10: No counterexample found. OP3 may be valid.
#11: No counterexample found. OP4 may be valid.
#12: No counterexample found. OP5 may be valid.

FIGURE: Alloy Output of a series of runs and the listed temporal properties

Final Remarks

1. We succesfully formally modelled S2Dedup using a Secure Deduplication scheme.
2. Would be interesting to implement other Secure Deduplication schemes, as they are much more complicated and would require rethinking our model.
3. Particularly Epoch based Secure Deduplication, as it would require hash changing values or using multiple hash functions.
4. In line with slide 12, it would be interesting to model a non injective hash function and see which properties still hold and how the system would need to be adapted to preserve ones that don't.

Final Remarks

1. We succesfully formally modelled S2Dedup using a Secure Deduplication scheme.
2. Would be interesting to implement other Secure Deduplication schemes, as they are much more complicated and would require rethinking our model.
3. Particularly Epoch based Secure Deduplication, as it would require hash changing values or using multiple hash functions.
4. In line with slide 12, it would be interesting to model a non injective hash function and see which properties still hold and how the system would need to be adapted to preserve ones that don't.

Final Remarks

1. We succesfully formally modelled S2Dedup using a Secure Deduplication scheme.
2. Would be interesting to implement other Secure Deduplication schemes, as they are much more complicated and would require rethinking our model.
3. Particularly Epoch based Secure Deduplication, as it would require hash changing values or using multiple hash functions.
4. In line with slide 12, it would be interesting to model a non injective hash function and see which properties still hold and how the system would need to be adapted to preserve ones that don't.

Final Remarks

1. We succesfully formally modelled S2Dedup using a Secure Deduplication scheme.
2. Would be interesting to implement other Secure Deduplication schemes, as they are much more complicated and would require rethinking our model.
3. Particularly Epoch based Secure Deduplication, as it would require hash changing values or using multiple hash functions.
4. In line with slide 12, it would be interesting to model a non injective hash function and see which properties still hold and how the system would need to be adapted to preserve ones that don't.

Bibliography I

Miranda, Mariana et al. (June 2021). "S2Dedup: SGX-enabled secure deduplication". en. In: *Proceedings of the 14th ACM International Conference on Systems and Storage*. ACM, pp. 1–12. ISBN: 978-1-4503-8398-1.

Paulo, João and José Pereira (June 2014). "A Survey and Classification of Storage Deduplication Systems". In: *ACM Comput. Surv.* 47.1. ISSN: 0360-0300.

Stallings, William (2010). *Cryptography and Network Security: Principles and Practice*. 5th. USA: Prentice Hall Press. ISBN: 0136097049.

Modelling Secure Multi User Deduplication - S2Dedup
Work Assignment
CSI

Bruno Gião PG57858

Tiago Teixeira PG57904

João Silva PG55618

June 24, 2025