Radboud University

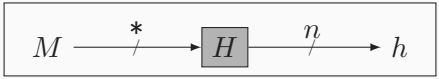
Generic Security of the SAFE API and Its Applications

Dmitry Khovratovich, Mario Marhuenda Beltrán, Bart Mennink

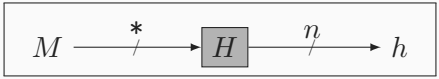
December 8, 2023

ESCADA

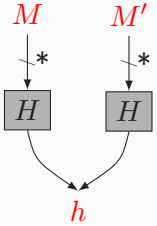
Cryptographic Hash Functions



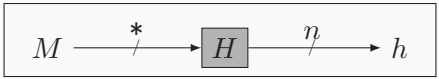
Cryptographic Hash Functions



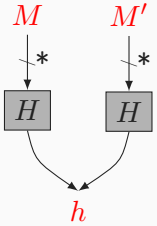
- **Collision resistant**

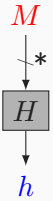


- **Preimage resistant**

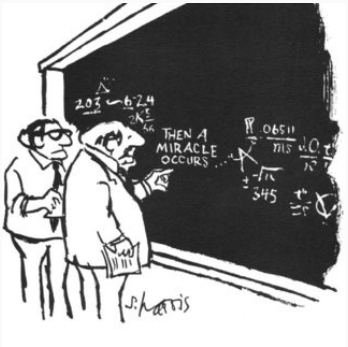


- **Collision resistant**
- **Preimage resistant**

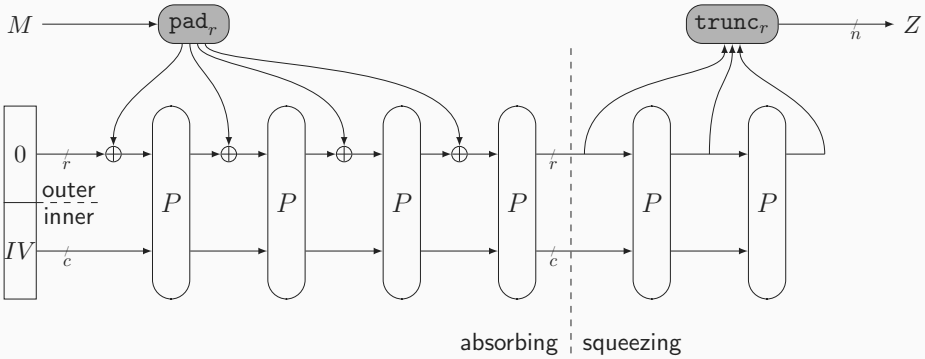




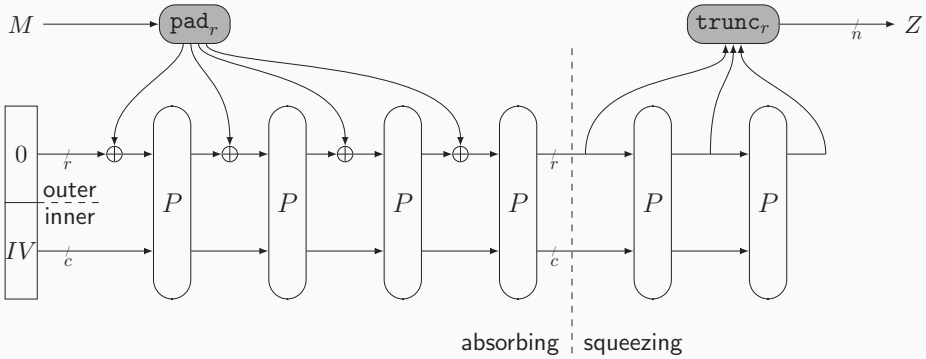
Mathematically easy,
but computationally hard!



Sponge Construction

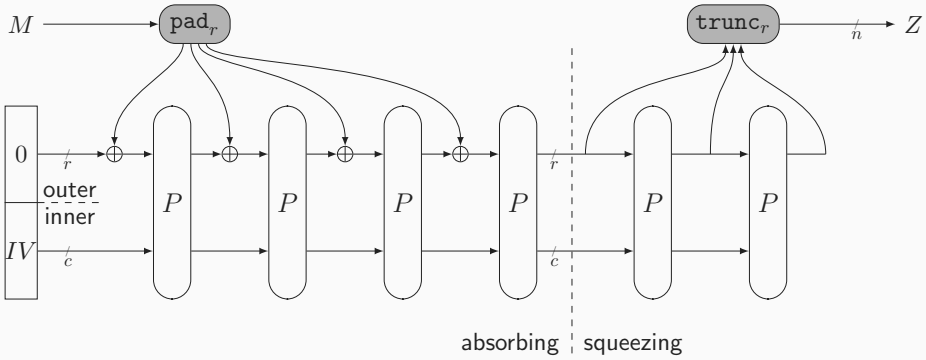


Sponge Construction



- P is b-bit permutation.
  - r is the rate.
  - c is the capacity.
  - $b = r + c$.

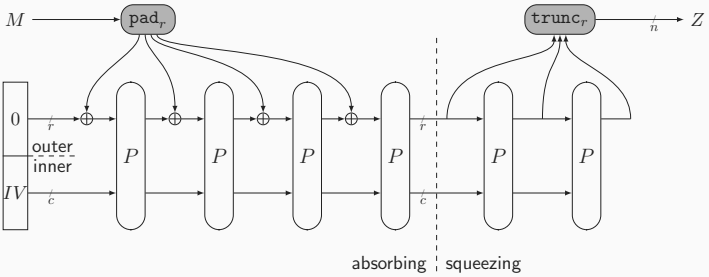
Sponge Construction



- P is b-bit permutation.
  - r is the rate.
  - c is the capacity.
  - $b = r + c$.
- **Security**: Behaves like RO up to $O(2^{c/2})$ queries [2, 3].

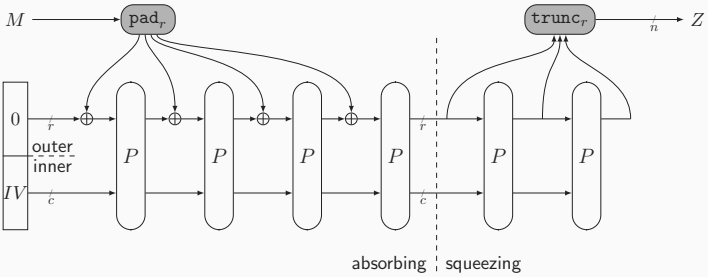
Our objective

A Problem



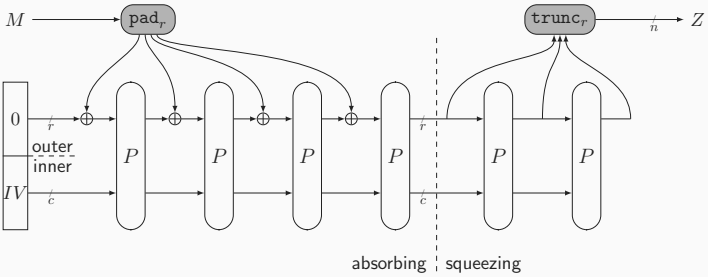
- **Padding** is necessary.

A Problem



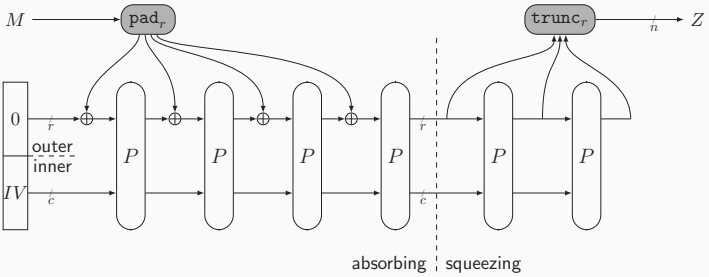
- **Padding** is necessary.
  - More absorption calls than if message would not be padded.

A Problem



- **Padding** is necessary.
  - More absorption calls than if message would not be padded.
    - More problematic in finite fields: **Inefficient**.
    - **Unnecessary evaluations** in some settings.

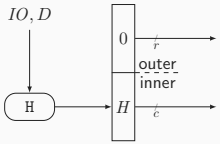
A Problem



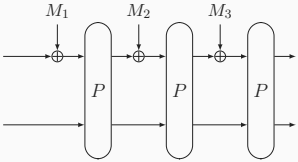
- **Padding** is necessary.
  - More absorption calls than if message would not be padded.
    - More problematic in finite fields: **Inefficient**.
    - **Unnecessary evaluations** in some settings.
- **Domain separation**.
  - We must make all the absorbs before any squeezing takes place.
  - **Inflexible scheme**.

A Solution: SAFE API [1]

Start

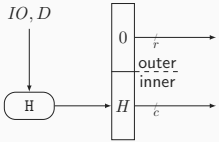


Absorb

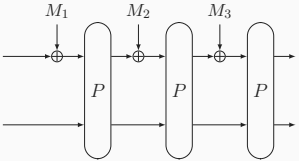


A Solution: SAFE API [1]

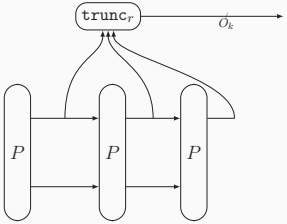
Start



Absorb



Squeeze



Finish

Features of SAFE API

- Fix **padding**:
  - Make the IV **dependent** on the message length.
  - Make the IV **dependent** on the absorb/squeeze order.

Features of SAFE API

- Fix **padding**:
  - Make the IV **dependent** on the message length.
  - Make the IV **dependent** on the absorb/squeeze order.
  - No more padding!

Features of SAFE API

- Fix **padding**:
  - Make the IV **dependent** on the message length.
  - Make the IV **dependent** on the absorb/squeeze order.
  - No more padding!
- IO fixes message **length**.
- Allows to alternate absorbs and squeezes.
- Include additional data in D.

Features of SAFE API

- Fix **padding**:
  - Make the IV **dependent** on the message length.
  - Make the IV **dependent** on the absorb/squeeze order.
  - No more padding!
- IO fixes message **length**.
- Allows to alternate absorbs and squeezes.
- Include additional data in D.

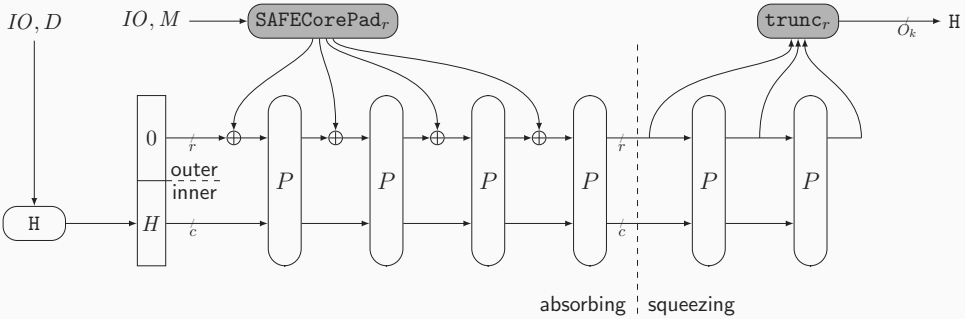
Security proofs of the sponge do **not** carry over.

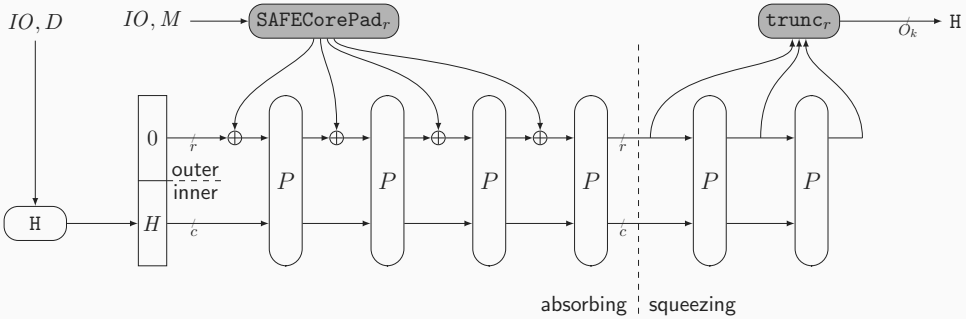
Features of SAFE API

- Fix **padding**:
  - Make the IV **dependent** on the message length.
  - Make the IV **dependent** on the absorb/squeeze order.
  - No more padding!
- IO fixes message **length**.
- Allows to alternate absorbs and squeezes.
- Include additional data in D.

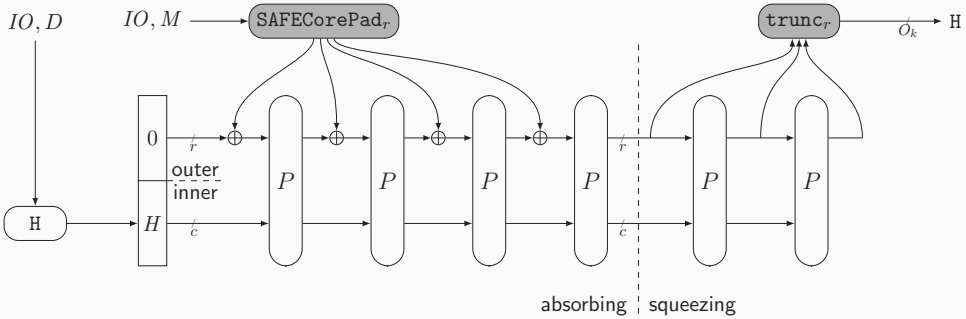
Security proofs of the sponge do **not** carry over.

- SAFECore: A variant of the **sponge**.
  - **Security**: Behaves like RO up to $O(2^{c/2})$ queries.

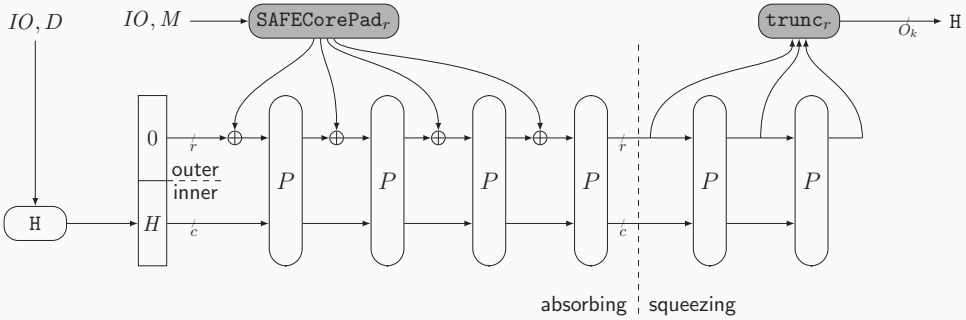

IO, D
IO, M
SAFECorePad_r
trunc_r
H
O_k
H
0
H
r
c
outer
inner
P
P
P
P
P
P
r
c
absorbing
squeezing



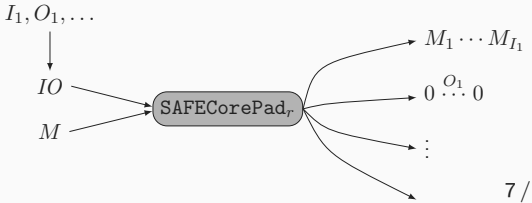
- **SAFECorePad**? You said no padding!



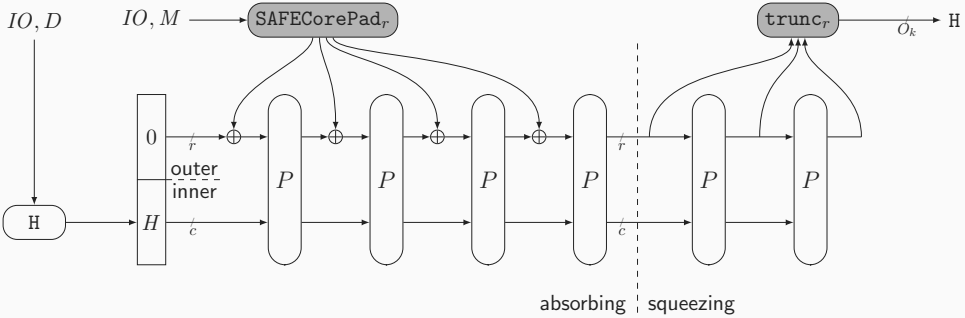
- **SAFECorePad**? You said no padding!
  - I was not lying...



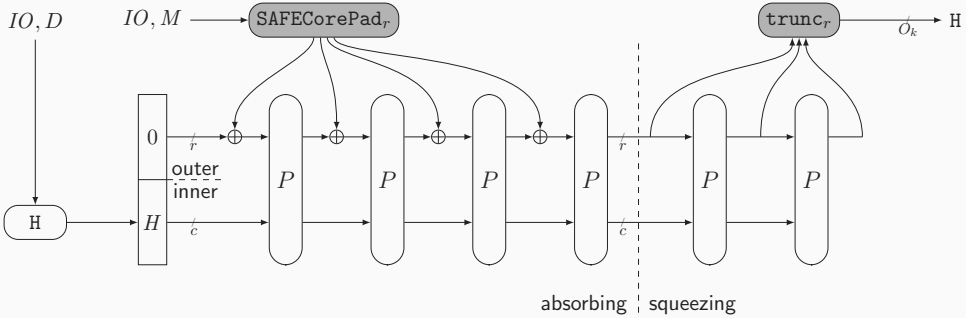
- **SAFECorePad**? You said no padding!
  - I was not lying...



SAFECore: Security

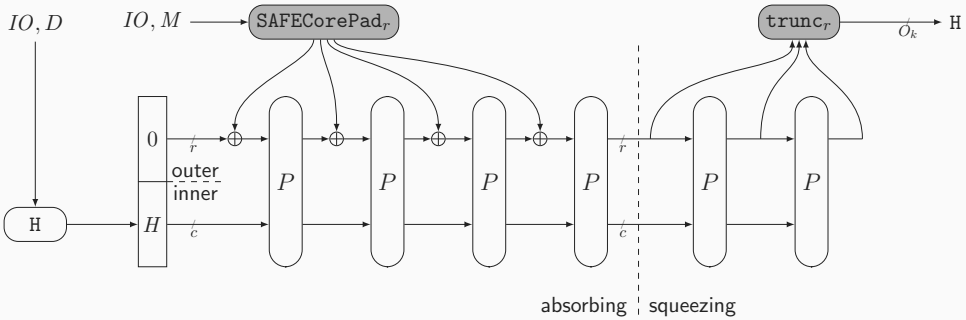


SAFECore: Security



- State of the art does not cover security of SAFECore.

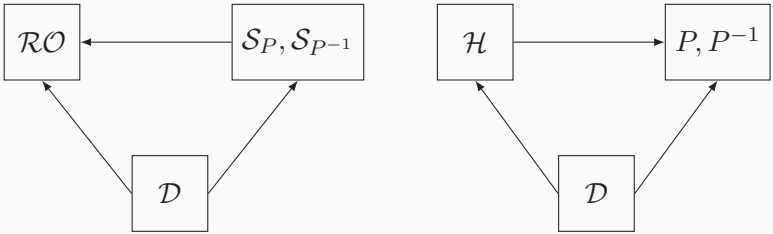
SAFECore: Security



- State of the art does not cover security of SAFECore.
- **Our contribution**: Thorough analysis of SAFE API. Previous state of the art: No proof for SAFE API.
  - We prove generic security of SAFECore ...
  - ... and apply it to SAFE API.

Security

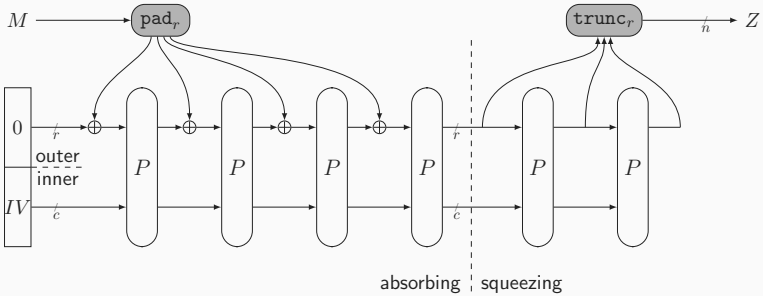
Indifferentiability framework



Ideal world

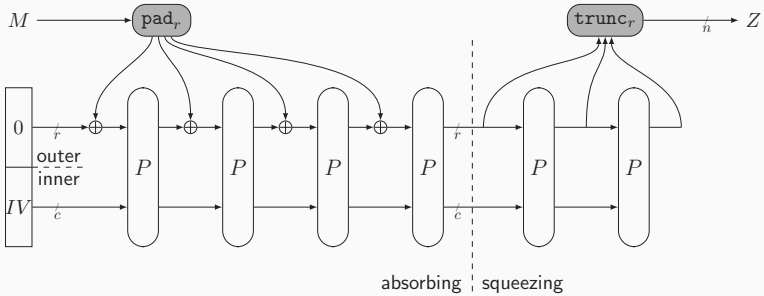
Real world

Indifferentiability of the Sponge



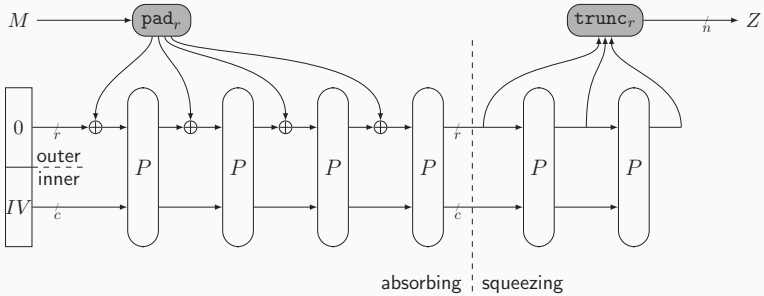
- General security bound [2, 3]: Indifferentiable from random oracle up to $O(2^{c/2})$ queries.

Indifferentiability of the Sponge



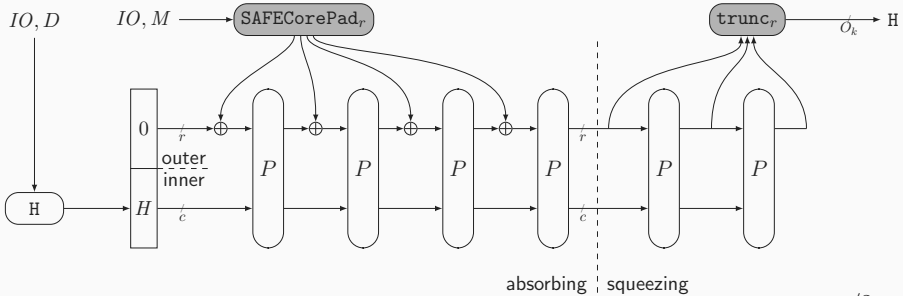
- General security bound [2, 3]: Indifferentiable from random oracle up to $O(2^{c/2})$ queries.
- **This result is tight**.
  - Collision in the inner part by querying P.
  - $\mathcal{D}$ can win the indifferentiability game with:

Indifferentiability of the Sponge



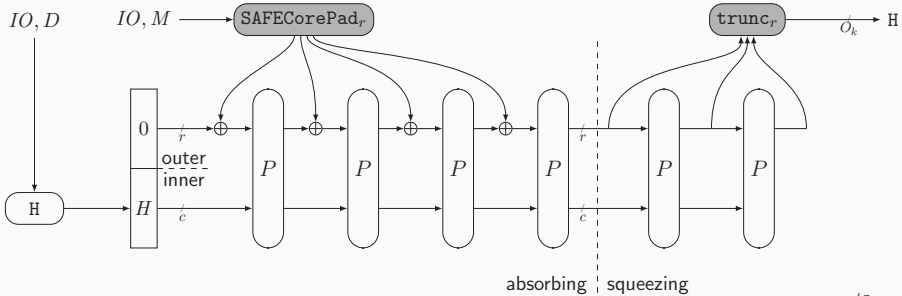
- General security bound [2, 3]: Indifferentiable from random oracle up to $O(2^{c/2})$ queries.
- **This result is tight**.
  - Collision in the inner part by querying P.
  - $\mathcal{D}$ can win the indifferentiability game with:
    - $O(2^{c/2})$ queries to P.

Indifferentiable of SAFECore



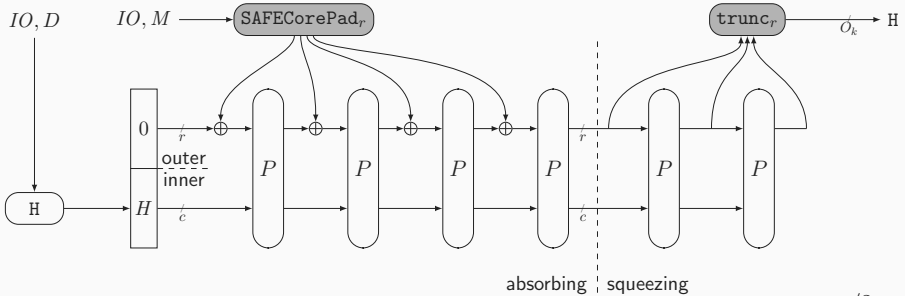
- General security bound: Indifferentiable from random oracle up to $O(2^{c/2})$ queries.
- **Same bound as in the sponge.**

Indifferentiable of SAFECore



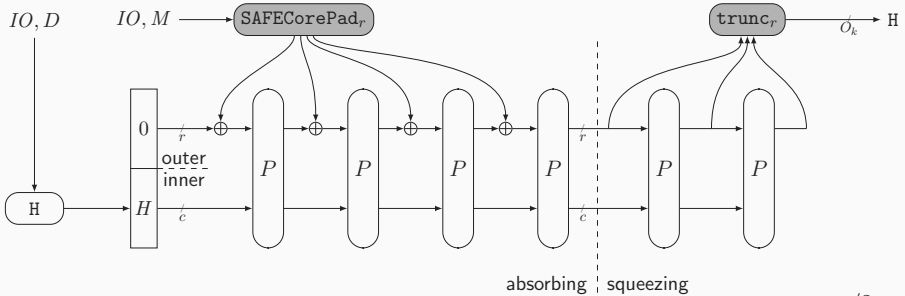
- General security bound: Indifferentiable from random oracle up to $O(2^{c/2})$ queries.
- **Same bound as in the sponge.**
  - Collision in the inner part by querying $\mathcal{P}$.
  - Now $\mathcal{D}$ can win the indifferentiability game.

Indifferentiable of SAFECore



- General security bound: Indifferentiable from random oracle up to $O(2^{c/2})$ queries.
- **Same bound as in the sponge.**
  - Collision in the inner part by querying $\mathcal{P}$.
  - Now $\mathcal{D}$ can win the indifferentiability game.
    - $O(2^{c/2})$ queries to P.

Indifferentiable of SAFECore



- General security bound: Indifferentiable from random oracle up to $O(2^{c/2})$ queries.
- **Same bound as in the sponge.**
  - Collision in the inner part by querying $\mathcal{P}$.
  - Now $\mathcal{D}$ can win the indifferentiability game.
    - $O(2^{c/2})$ queries to P.
  - **A new attack**: Collision in the inner part by querying $\mathcal{H}$.
  - We lost nothing because we already had this bound in the sponge.

Applications

Applications of SAFE API

- Plain hashing.
- Commitment schemes.
- Interactive protocols.
- Merkle trees.
- Zero Knowledge proofs: SNARKs.
- Lattice cryptography.
- ZKVMs.
- Verifiable encryption.

Applications of SAFE API: Commitment Scheme

- Suppose you want to commit to a l-tuple: $(X_1, \ldots, X_l) \in \mathbb{F}_q^l$.

Applications of SAFE API: Commitment Scheme

- Suppose you want to commit to a l-tuple: $(X_1, \ldots, X_l) \in \mathbb{F}_q^l$.
  1: $IO \leftarrow (l, \mu)$
  2: $D \leftarrow \varnothing$

Applications of SAFE API: Commitment Scheme

- Suppose you want to commit to a l-tuple: $(X_1, \ldots, X_l) \in \mathbb{F}_q^l$.
  1: $IO \leftarrow (l, \mu)$
  2: $D \leftarrow \varnothing$
  3: $\texttt{START}(IO, D)$

Applications of SAFE API: Commitment Scheme

- Suppose you want to commit to a l-tuple: $(X_1, \ldots, X_l) \in \mathbb{F}_q^l$.
  1: $IO \leftarrow (l, \mu)$
  2: $D \leftarrow \varnothing$
  3: $\texttt{START}(IO, D)$
  4: $\texttt{ABSORB}(\ell \cdot d + 1, X_1 \parallel X_2 \parallel \cdots \parallel X_\ell \parallel R)$, $R \xleftarrow{\$} \mathbb{F}_q$

Applications of SAFE API: Commitment Scheme

- Suppose you want to commit to a l-tuple: $(X_1, \ldots, X_l) \in \mathbb{F}_q^l$.
  1: $IO \leftarrow (l, \mu)$
  2: $D \leftarrow \varnothing$
  3: $\texttt{START}(IO, D)$
  4: $\texttt{ABSORB}(\ell \cdot d + 1, X_1 \parallel X_2 \parallel \cdots \parallel X_\ell \parallel R)$, $R \xleftarrow{\$} \mathbb{F}_q$
  5: $Z \leftarrow \texttt{SQUEEZE}(\mu)$

Applications of SAFE API: Commitment Scheme

- Suppose you want to commit to a l-tuple: $(X_1, \ldots, X_l) \in \mathbb{F}_q^l$.
  1: $IO \leftarrow (l, \mu)$
  2: $D \leftarrow \varnothing$
  3: $\texttt{START}(IO, D)$
  4: $\texttt{ABSORB}(\ell \cdot d + 1, X_1 \parallel X_2 \parallel \cdots \parallel X_\ell \parallel R)$, $R \xleftarrow{\$} \mathbb{F}_q$
  5: $Z \leftarrow \texttt{SQUEEZE}(\mu)$
  6: $\texttt{FINISH}()$
  7: **return** Z

Applications of SAFE API: Commitment Scheme

- Suppose you want to commit to a l-tuple: $(X_1, \ldots, X_l) \in \mathbb{F}_q^l$.
  1: $IO \leftarrow (l, \mu)$
  2: $D \leftarrow \varnothing$
  3: $\texttt{START}(IO, D)$
  4: $\texttt{ABSORB}(\ell \cdot d + 1, X_1 \parallel X_2 \parallel \cdots \parallel X_\ell \parallel R)$, $R \xleftarrow{\$} \mathbb{F}_q$
  5: $Z \leftarrow \texttt{SQUEEZE}(\mu)$
  6: $\texttt{FINISH}()$
  7: **return** Z

- **Translation** to SAFECore:

$$Z \leftarrow \texttt{SAFECore}((l, \mu), \varnothing, X_1 \| X_2 \| \cdots \| X_\ell \| R)\,.$$

Applications of SAFE API: Commitment Scheme

- Suppose you want to commit to a l-tuple: $(X_1, \ldots, X_l) \in \mathbb{F}_q^l$.
  1: $IO \leftarrow (l, \mu)$
  2: $D \leftarrow \varnothing$
  3: $\texttt{START}(IO, D)$
  4: $\texttt{ABSORB}(\ell \cdot d + 1, X_1 \parallel X_2 \parallel \cdots \parallel X_\ell \parallel R)$, $R \xleftarrow{\$} \mathbb{F}_q$
  5: $Z \leftarrow \texttt{SQUEEZE}(\mu)$
  6: $\texttt{FINISH}()$
  7: **return** Z

- **Translation** to SAFECore:

$$Z \leftarrow \texttt{SAFECore}((l, \mu), \varnothing, X_1 \| X_2 \| \cdots \| X_\ell \| R)\,.$$

- Generic security of SAFECore implies security of commitment scheme.

Conclusion

- **Result**: Formal generic analysis of SAFE API.
- Allows for more efficient hashing in finite fields.
  - Requires the use of a **another hash function**.

Conclusion

- **Result**: Formal generic analysis of SAFE API.
- Allows for more efficient hashing in finite fields.
  - Requires the use of a **another hash function**.
- Generic security bound is **the same** as normal sponge.

Thank you for your attention!

Bibliography

- Aumasson, J., Khovratovich, D., Quine, P.: SAFE (Sponge API for Field Elements) – A Toolbox for ZK Hash Applications (2022), https://safe-hash.dev/
- Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: On the Indifferentiability of the Sponge Construction. In: Smart, N.P. (ed.) Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings. Lecture Notes in Computer Science, vol. 4965, pp. 181–197. Springer (2008), https://doi.org/10.1007/978-3-540-78967-3_11
- Naito, Y., Ohta, K.: Improved Indifferentiable Security Analysis of PHOTON. In: Abdalla, M., Prisco, R.D. (eds.) Security and Cryptography for Networks - 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8642, pp. 340–357. Springer (2014), https://doi.org/10.1007/978-3-319-10879-7_20