STATE-RECOVERY ANALYSIS OF SPRITZ

Ralph Ankele[1] **Stefan Kölbl**[2] Christian Rechberger[2]

August 25, 2015

[1]RHUL, Royal Holloway University of London, United Kingdom

[2]DTU Compute, Technical University of Denmark, Denmark

RC4 AND TLS

RC4

RC4

- Stream Cipher
- Designed in 1987 by Ron Rivest
- Fast in Software
- Used in TLS (Transport Layer Security)

Produces key stream

$$z = z_0||z_1||\ldots||z_k \tag{1}$$

RC4

Output bytes z_i of RC4 are biased

- $\Pr[z_2 = 0] \approx \frac{1}{128}$ [FMS01]
- Distribution of z_1 [Mir02]
- $\Pr[z_l = 256 - l] \geq \frac{1}{256} + \frac{1}{256^2}$ [GMPS11]

[1] https://www.rc4nomore.com

RC4

Output bytes z_i of RC4 are biased

- $\Pr[z_2 = 0] \approx \frac{1}{128}$ [FMS01]
- Distribution of z_1 [Mir02]
- $\Pr[z_l = 256 - l] \geq \frac{1}{256} + \frac{1}{256^2}$ [GMPS11]

Attack on TLS using RC4

- Plaintext recovery for TLS using RC4 [ABP+13]
- Needs around 2^{30} sessions.

[1] https://www.rc4nomore.com

RC4

Output bytes z_i of RC4 are biased

- $\Pr[z_2 = 0] \approx \frac{1}{128}$ [FMS01]
- Distribution of z_1 [Mir02]
- $\Pr[z_l = 256 - l] \geq \frac{1}{256} + \frac{1}{256^2}$ [GMPS11]

Attack on TLS using RC4

- Plaintext recovery for TLS using RC4 [ABP+13]
- Needs around 2^{30} sessions.

Usenix'15

- Break WPA-TKIP and decrypt cookies in 75 hours [MF15] [1].
- Password Recovery TLS [CPdMT15]

[1] https://www.rc4nomore.com

RC4

RC4 should **NOT** be used anymore!

- In July 2014, 10% of servers do not support RC4.
- In July 2015, 40% of servers do not support RC4[2].
- IETF Draft to remove RC4 from TLS RFC7465.

[2] SSL Pulse, July 07, 2015


RC4

SPRITZ

Spritz

A redesign of RC4 by Ron Rivest and Jacob C. N. Schuldt

- Avoid statistical weakness of RC4.
- Update function chosen using extensive computations.
- Uses a Sponge-like construction.

RC4

```
i = i + 1
j = j + S[i]
SWAP(S[i], S[j])
z = S[S[i] + S[j]]
return z
```

Spritz

```
i = i + w
j = k + S[j + S[i]]
k = i + k + S[j]
SWAP(S[i], S[j])
z = S[j + S[i + S[z + k]]]
return z
```

Table 1: Performance of stream ciphers for Software.

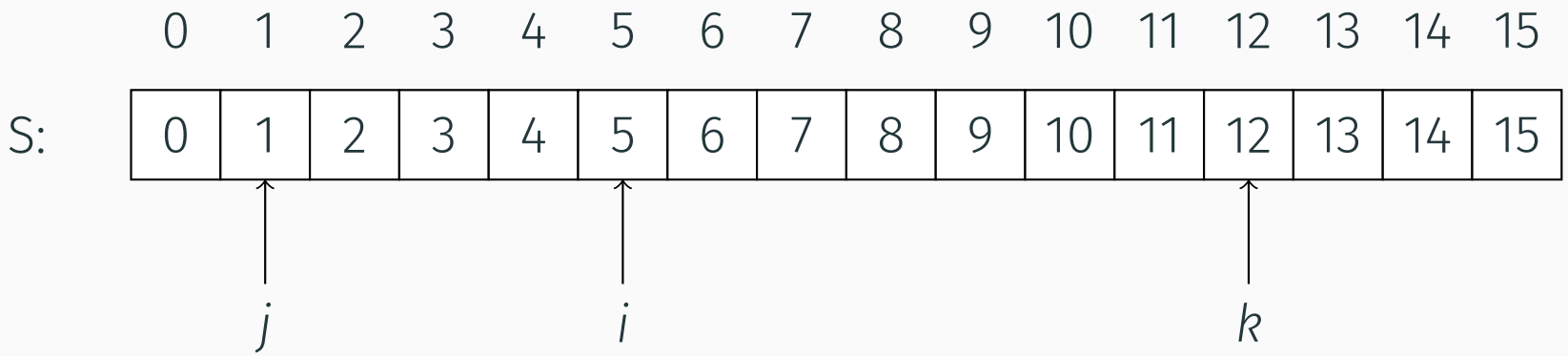
Cipher	Long Msg.	Short Msg.[3]
RC4	293 MB/s	142 MB/s
Spritz	95 MB/s	32 MB/s
Salsa20	296 MB/s	268 MB/s
AES-CTR	152 MB/s	146 MB/s

Spritz implementation is not optimized in this comparison.

[3] 16 byte key, 512 bytes

Internal structure:

- Six registers: i, j, k, w, z and a.
- Permutation:



Spritz

Sponge-like construction supports many applications

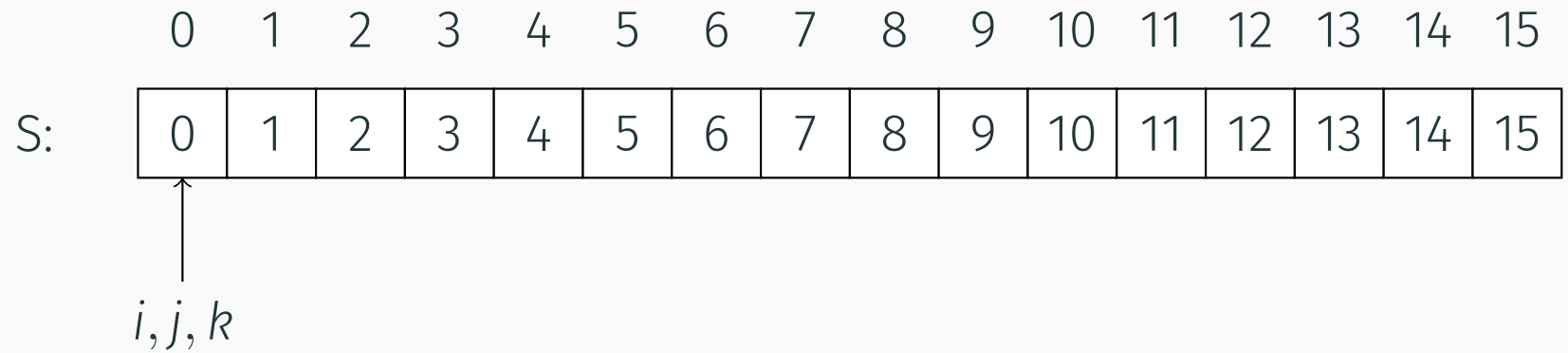
- Encryption
- Hashing
- MAC

Example Encryption:

```
Encrypt(M, K):
    InitializeState()
    Absorb(K)
    C = M + Squeeze(M.length)
```

InitializeState()

- First all registers are initialized: $i = j = k = z = a = 0, w = 1$
- Initialize Permutation:



Spritz

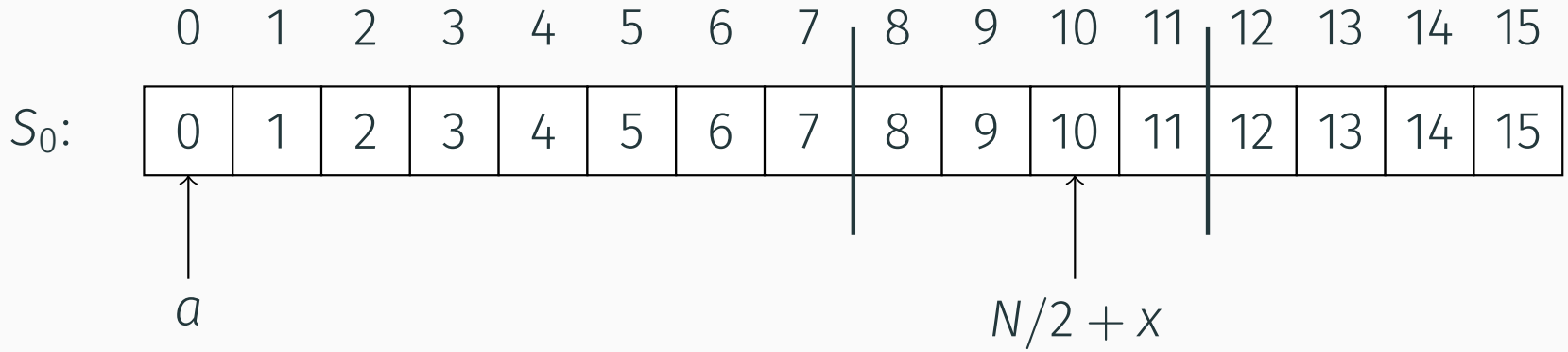
Absorb(x) using $x = 2||0||1||2$

- $Swap(S[a], S[N/2 + x])$
- $a = a + 1$

Absorb(x) using $x = 2||0||1||2$

- $Swap(S[a], S[N/2 + x])$
- $a = a + 1$

Example:

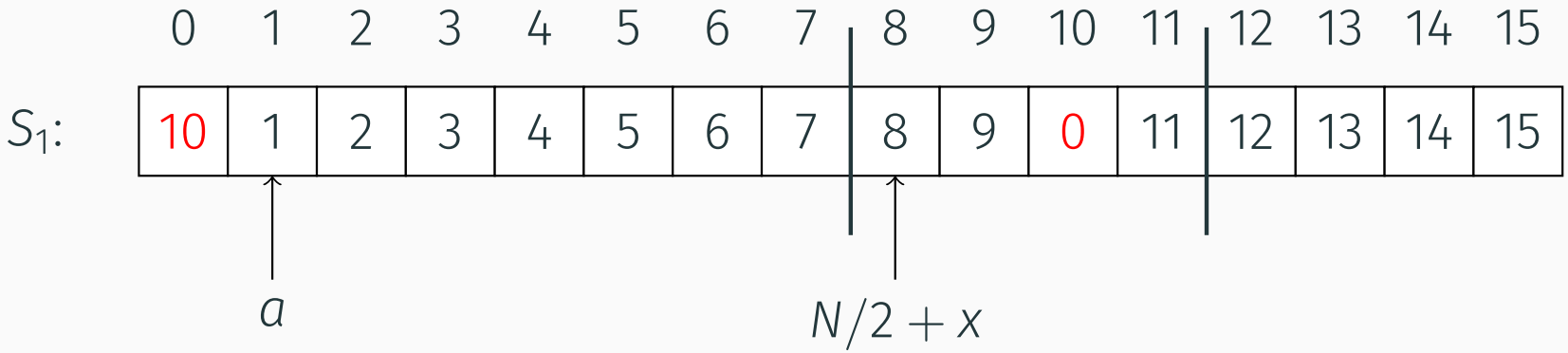


Absorb **2**

Absorb(x) using $x = 2||0||1||2$

- $Swap(S[a], S[N/2 + x])$
- $a = a + 1$

Example:

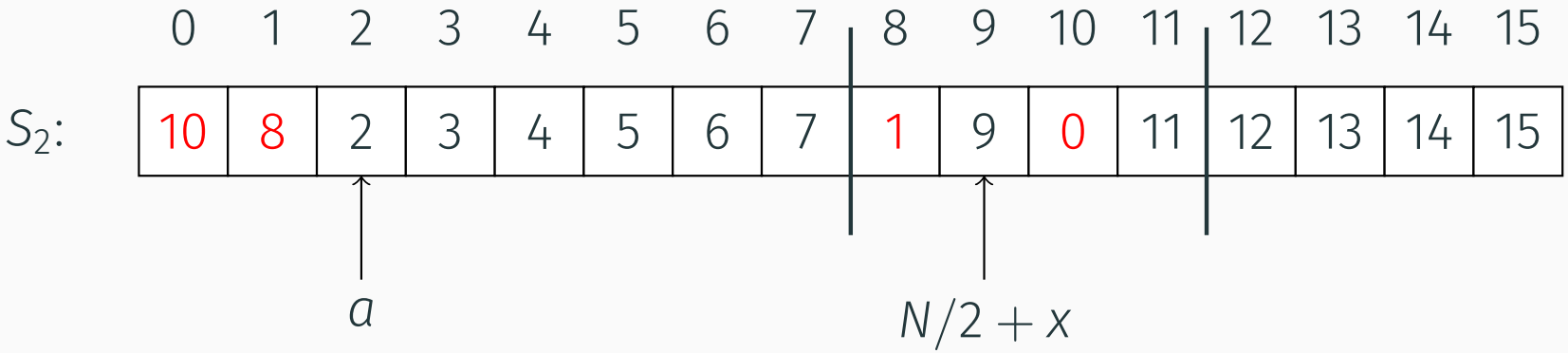


Absorb `0`

Absorb(x) using $x = 2||0||1||2$

- $Swap(S[a], S[N/2 + x])$
- $a = a + 1$

Example:

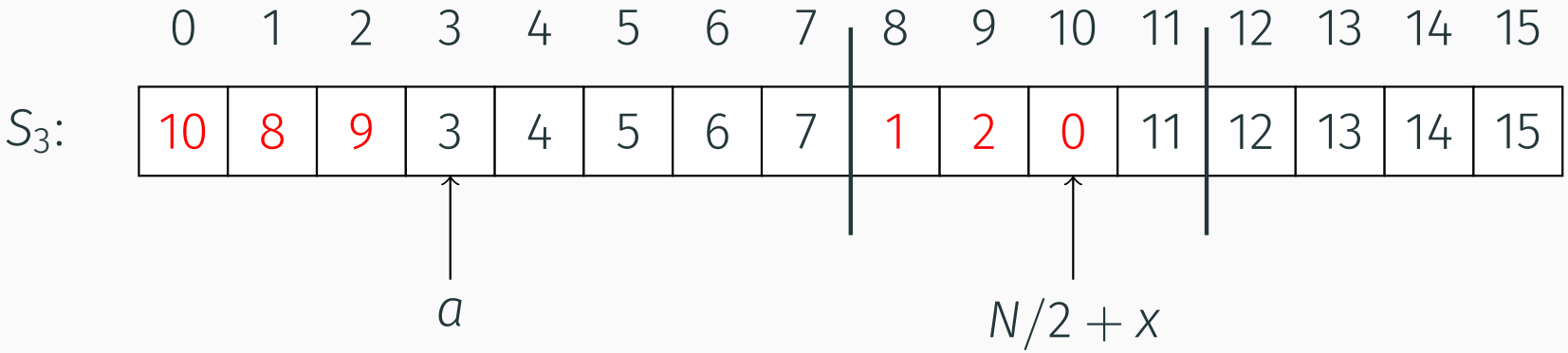


Absorb **1**

Absorb(x) using $x = 2||0||1||2$

- $Swap(S[a], S[N/2 + x])$
- $a = a + 1$

Example:

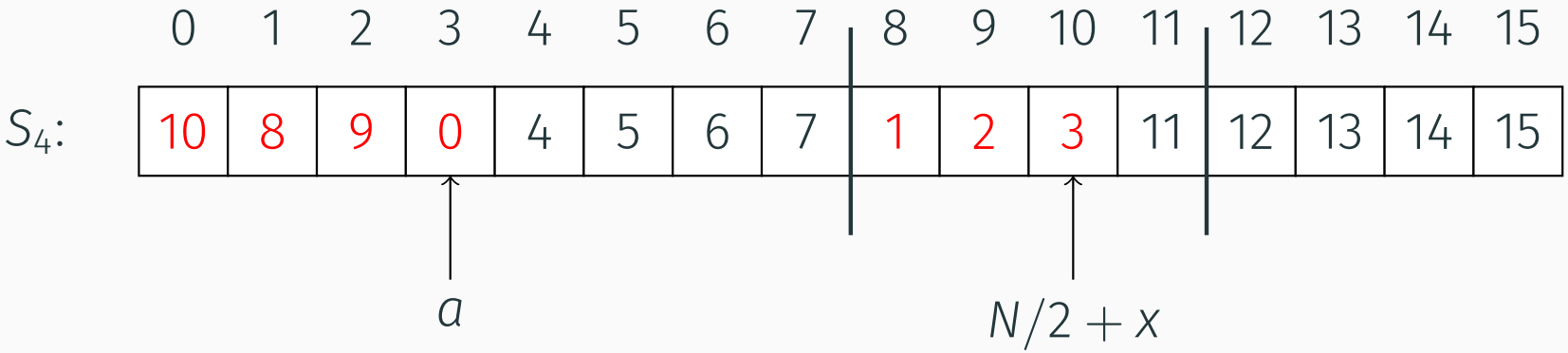


Absorb 2

Absorb(x) using $x = 2||0||1||2$

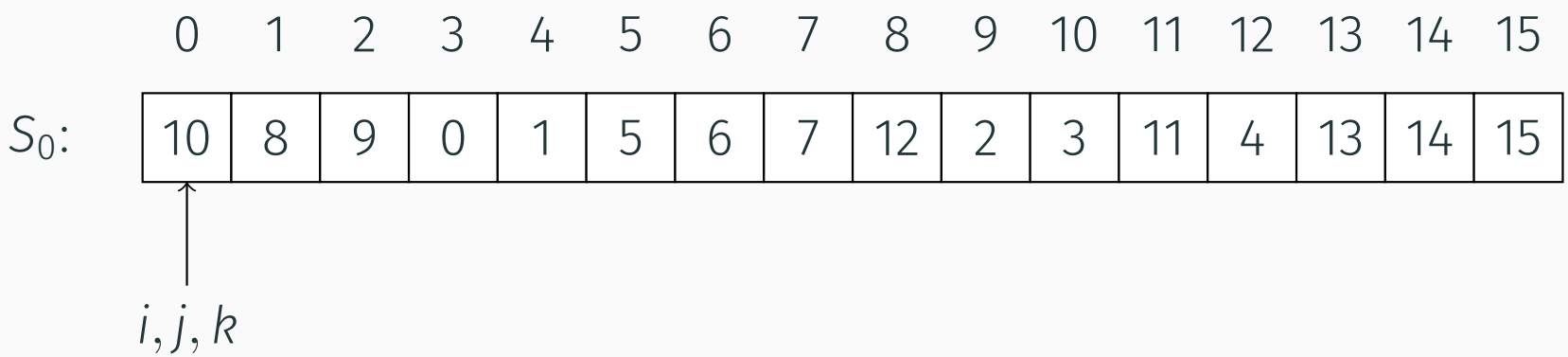
- $Swap(S[a], S[N/2 + x])$
- $a = a + 1$

Example:



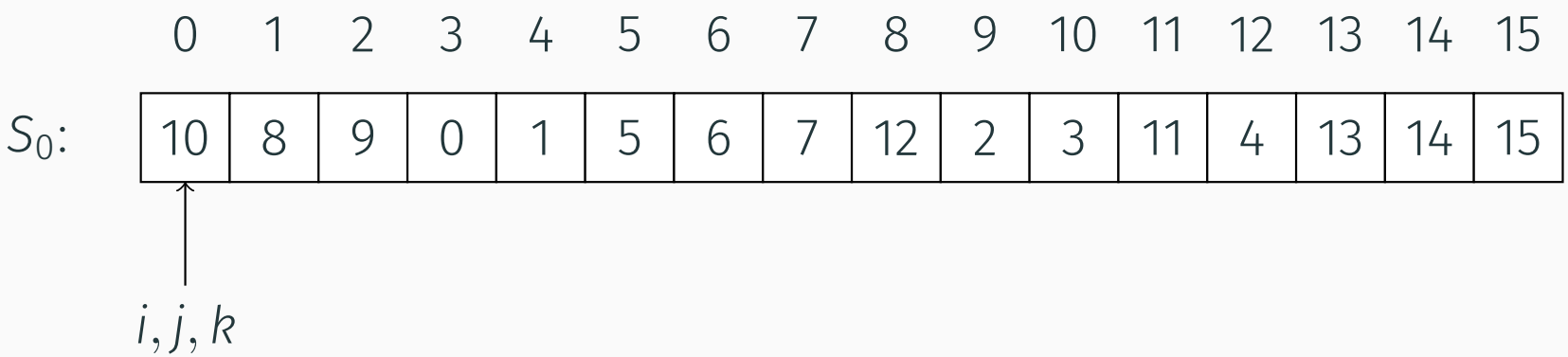
Spritz

```
Update:
    i = i + w
    j = k + S[j + S[i]]
    k = i + k + S[j]
    SWAP(S[i], S[j])
    z = S[j + S[i + S[z + k]]]
    return z
```



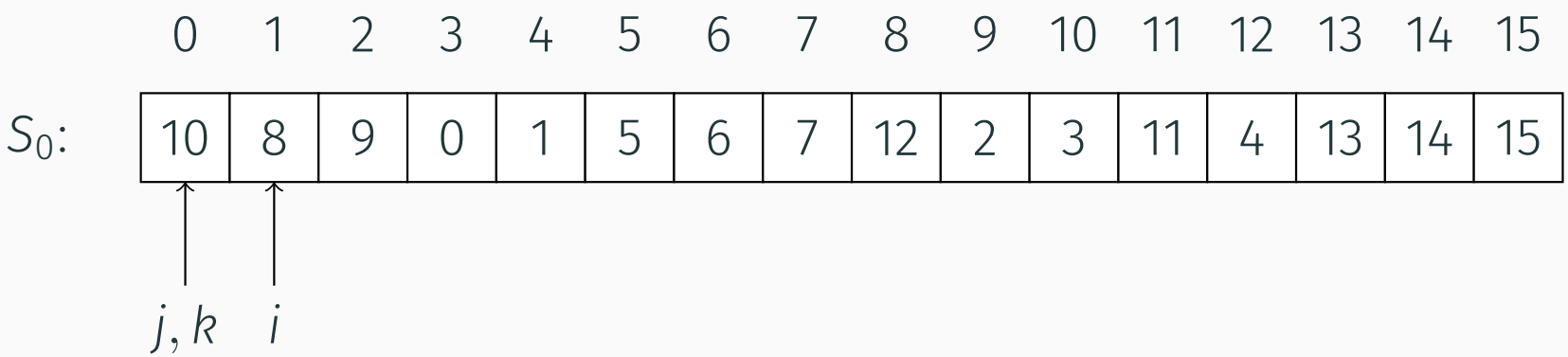
Spritz

```
Update:
    i = i + w
    j = k + S[j + S[i]]
    k = i + k + S[j]
    SWAP(S[i], S[j])
    z = S[j + S[i + S[z + k]]]
    return z
```



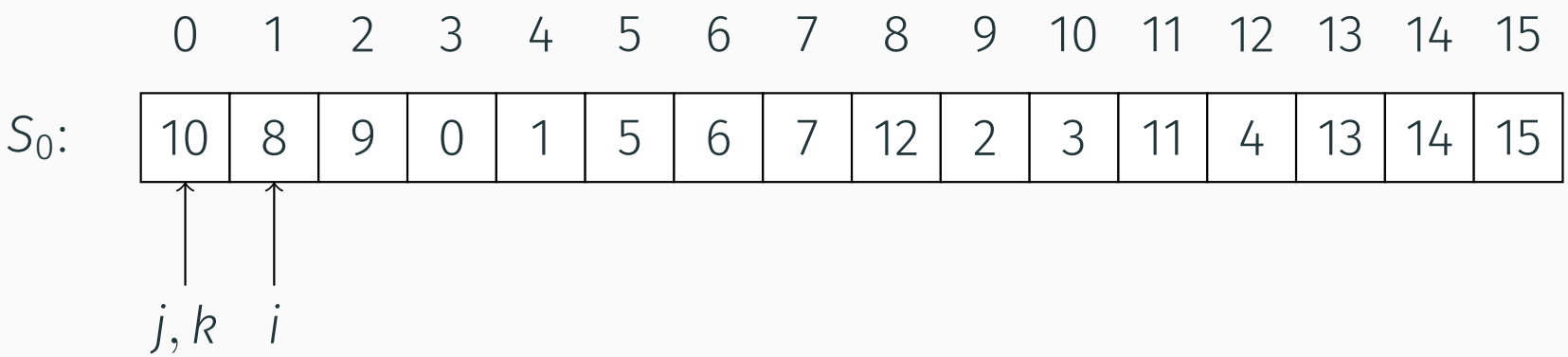
Spritz

```
Update:
    i = i + w
    j = k + S[j + S[i]]
    k = i + k + S[j]
    SWAP(S[i], S[j])
    z = S[j + S[i + S[z + k]]]
    return z
```



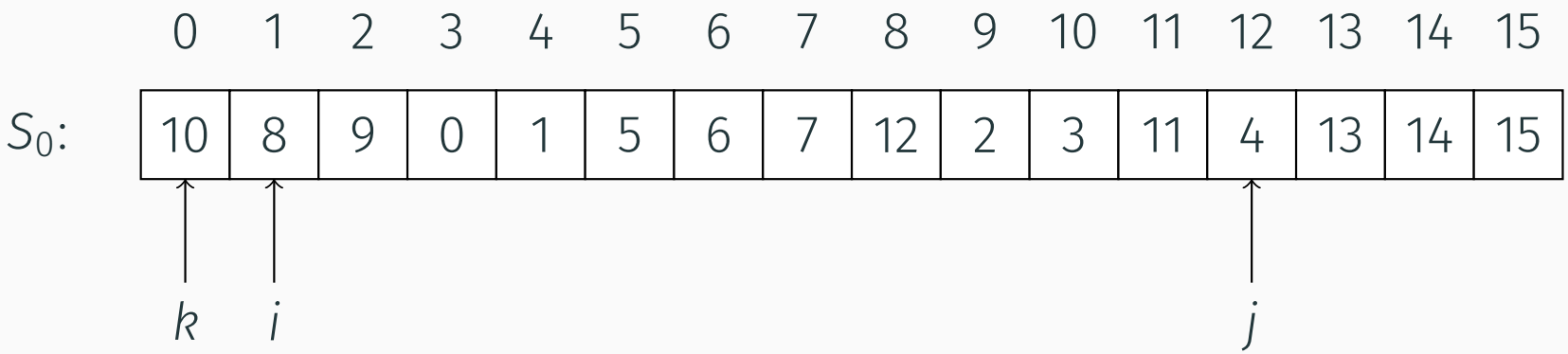
Spritz

```
Update:
    i = i + w
    j = k + S[j + S[i]]
    k = i + k + S[j]
    SWAP(S[i], S[j])
    z = S[j + S[i + S[z + k]]]
    return z
```



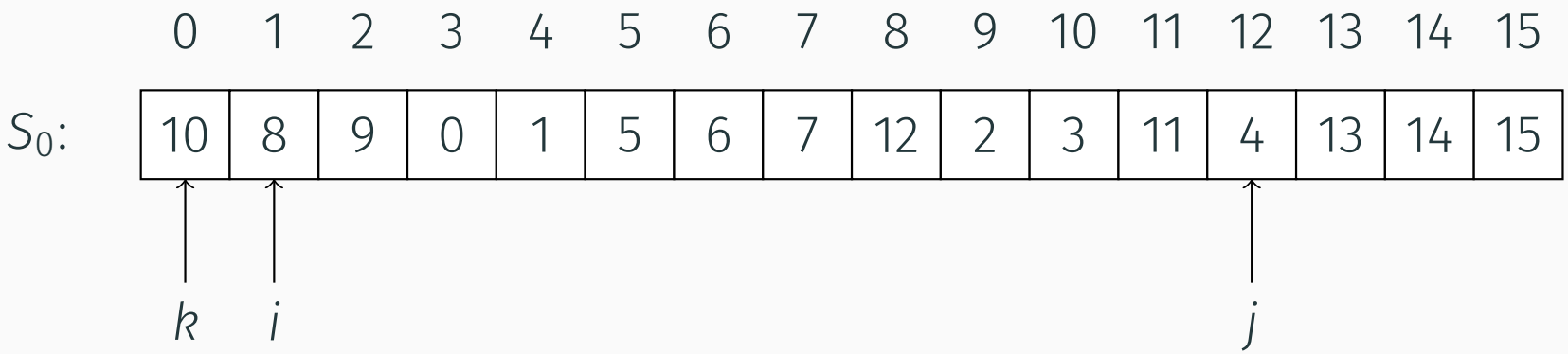
Spritz

```
Update:
    i = i + w
    j = k + S[j + S[i]]
    k = i + k + S[j]
    SWAP(S[i], S[j])
    z = S[j + S[i + S[z + k]]]
    return z
```



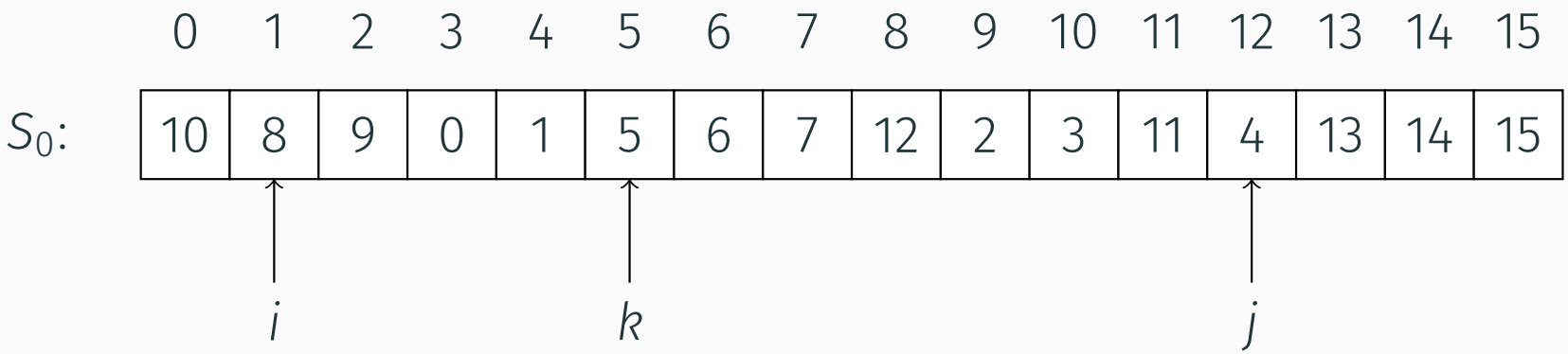
Spritz

```
Update:
    i = i + w
    j = k + S[j + S[i]]
    k = i + k + S[j]
    SWAP(S[i], S[j])
    z = S[j + S[i + S[z + k]]]
    return z
```



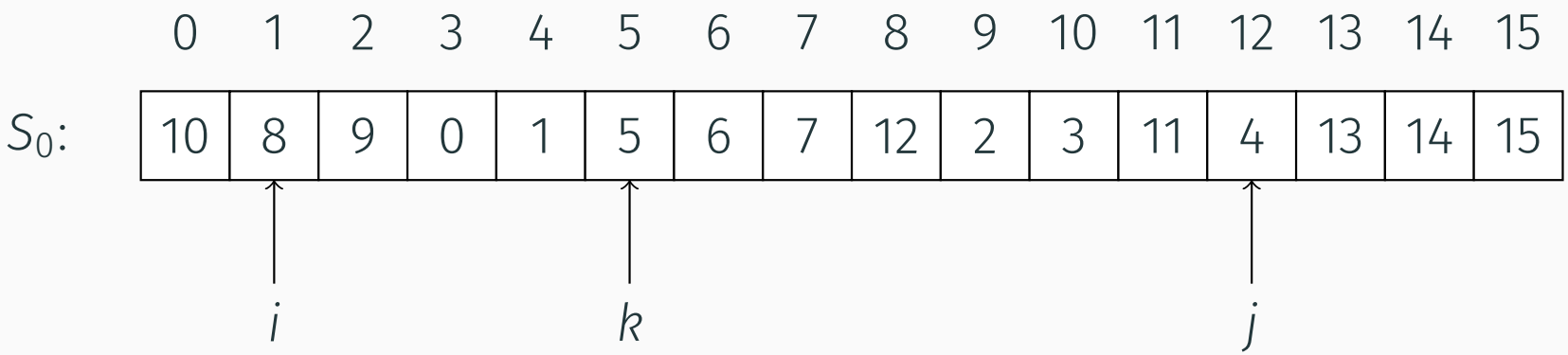
Spritz

```
Update:
    i = i + w
    j = k + S[j + S[i]]
    k = i + k + S[j]
    SWAP(S[i], S[j])
    z = S[j + S[i + S[z + k]]]
    return z
```



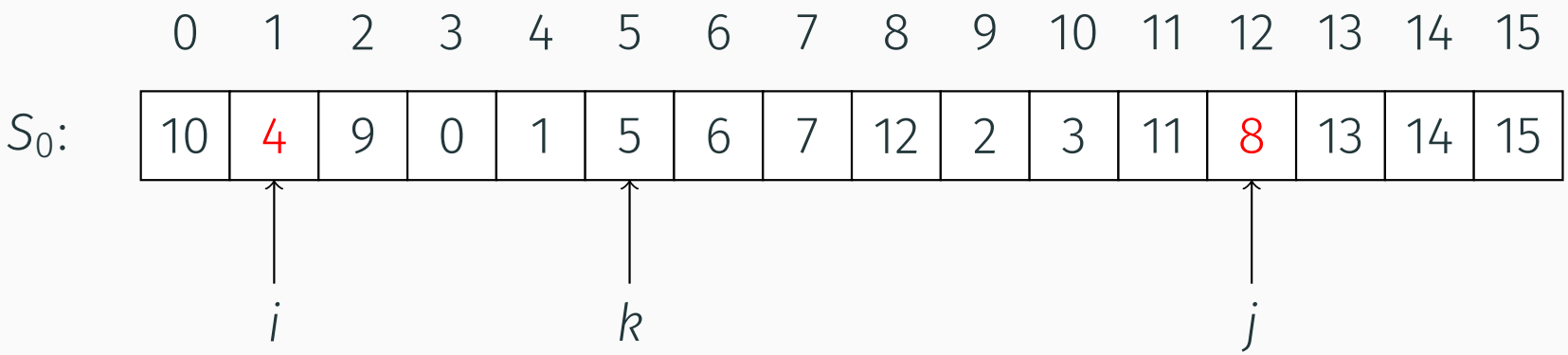
Spritz

```
Update:
    i = i + w
    j = k + S[j + S[i]]
    k = i + k + S[j]
    SWAP(S[i], S[j])
    z = S[j + S[i + S[z + k]]]
    return z
```



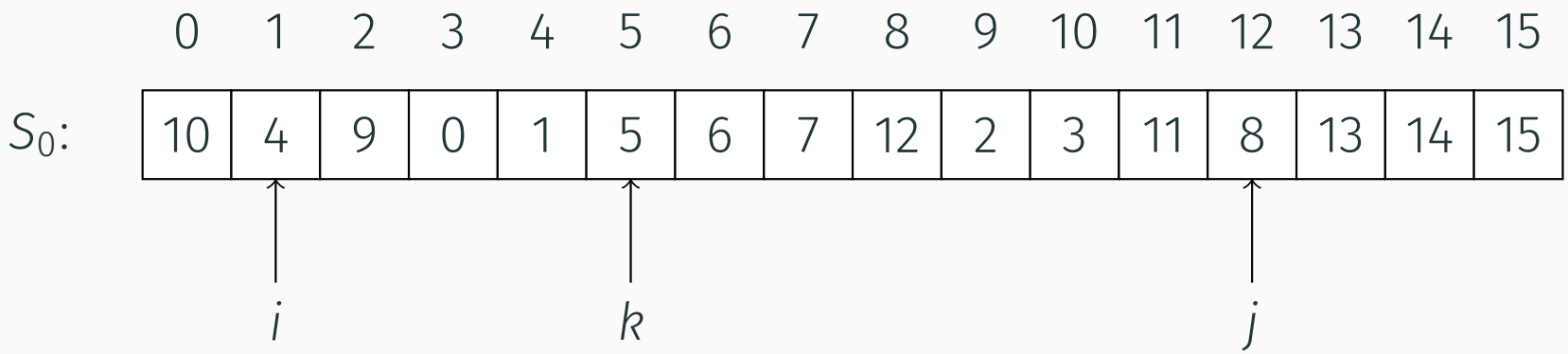
Spritz

```
Update:
    i = i + w
    j = k + S[j + S[i]]
    k = i + k + S[j]
    SWAP(S[i], S[j])
    z = S[j + S[i + S[z + k]]]
    return z
```



Spritz

```
Update:
    i = i + w
    j = k + S[j + S[i]]
    k = i + k + S[j]
    SWAP(S[i], S[j])
    z = S[j + S[i + S[z + k]]]
    return z
```



Keystream output: $z = S[12 + S[1 + S[0 + 5]]] = \mathbf{10}$

Spritz

Spritz integrates the common practice of throwing away bytes of the keystream in the design.

```
Shuffle:
    Whip(2N)
    Crush()
    Whip(2N)
    Crush()
    Whip(2N)
```

- Whip: Calls Update and throws away output bytes.
- Crush: A simple many-to-one mapping.

STATE RECOVERY

State Recovery

State Recovery for RC4

- Allows to predict key stream
- Recover the initial state
- Different techniques and optimizations [KMP+98] [MK08].
- Complexity is high

State Recovery

Cipher	Permutation size	Time	Reference
RC4	32	2^{53}	[KMP+98]
RC4	64	2^{60}	[MK08]
RC4	128	2^{113}	[MK08]
RC4	256	2^{241}	[MK08]

How do these attacks perform in the case of Spritz?

State Recovery

We adapt the method presented in [KMP+98] for Spritz

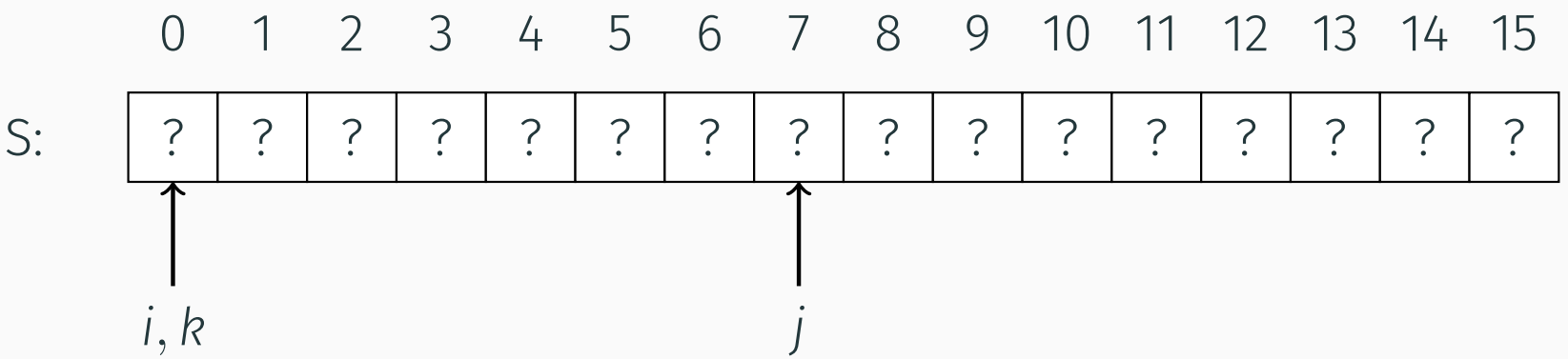
- We observe the sequence of output bytes $z_0, z_1, \ldots$
- Simulate `Update` as long as we know the inputs.
- Guess any unknown values.

Main differences to RC4

- Need to guess more values of S in each step.
- Can only recover state up to last call of Crush().

State Recovery

Registers: $i_0 = 0, w_0 = 1, j_0 = 7, k_0 = 0, z_0 = 0$:



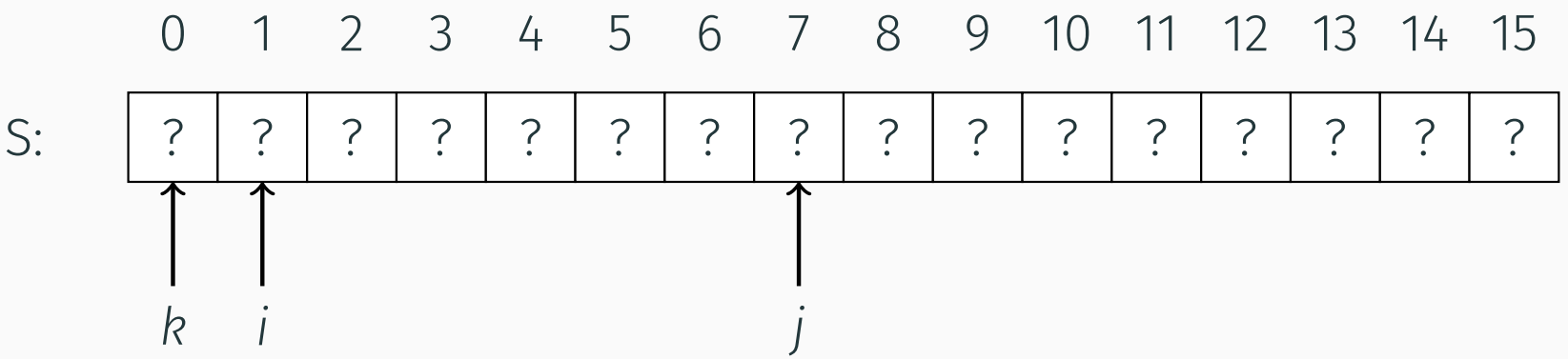

```
Update:
    i = i + w
    j = k + S[j + S[i]]
    k = i + k + S[j]
    SWAP(S[i], S[j])
    z = S[j + S[i + S[z + k]]]
    return z
```

Simulate Update:

- Guess $S[i]$ and $S[j + S[i]]$
- Guess $S[j]$
- Guess $S[z + k]$ and $S[i + S[z + k]]$
- $S[j + S[i + S[z + k]]] = z_1 = 12$

State Recovery

Registers: $i_0 = 0, w_0 = 1, j_0 = 7, k_0 = 0, z_0 = 0$:



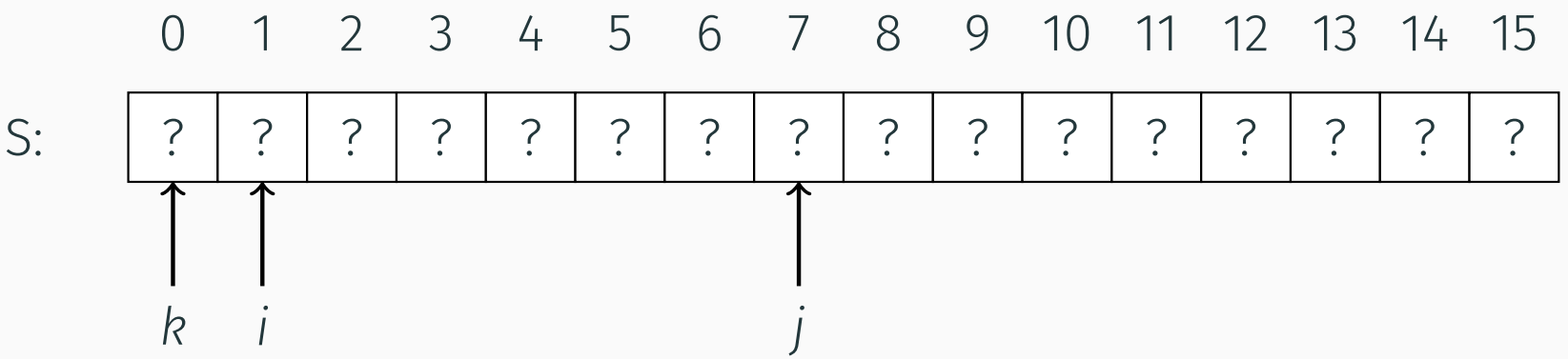

```
Update:
    i = i + w
    j = k + S[j + S[i]]
    k = i + k + S[j]
    SWAP(S[i], S[j])
    z = S[j + S[i + S[z + k]]]
    return z
```

Simulate Update:

- Guess $S[i]$ and $S[j + S[i]]$
- Guess $S[j]$
- Guess $S[z + k]$ and $S[i + S[z + k]]$
- $S[j + S[i + S[z + k]]] = z_1 = 12$

State Recovery

Registers: $i_0 = 0, w_0 = 1, j_0 = 7, k_0 = 0, z_0 = 0$:



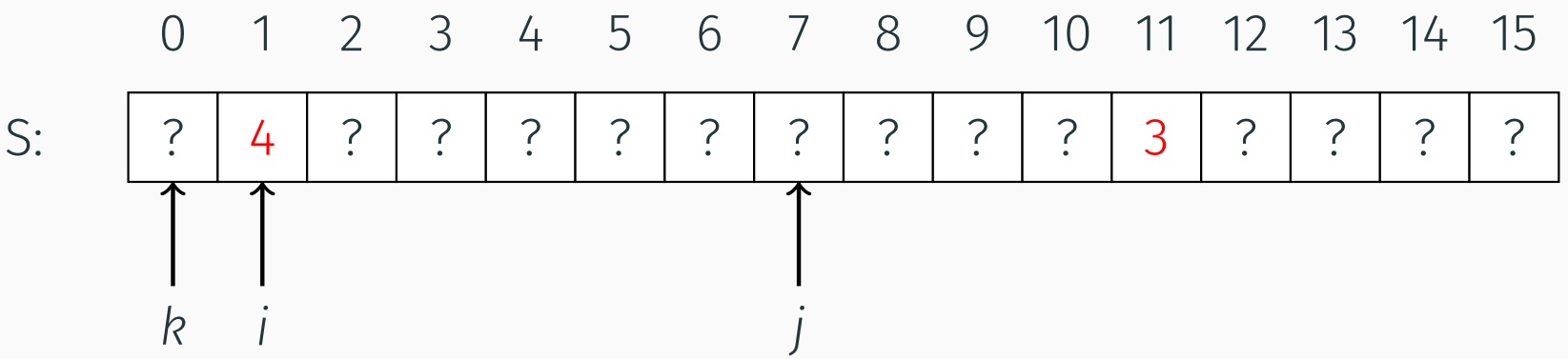

```
Update:
    i = i + w
    j = k + S[j + S[i]]
    k = i + k + S[j]
    SWAP(S[i], S[j])
    z = S[j + S[i + S[z + k]]]
    return z
```

Simulate Update:

- Guess $S[i]$ and $S[j + S[i]]$
- Guess $S[j]$
- Guess $S[z + k]$ and $S[i + S[z + k]]$
- $S[j + S[i + S[z + k]]] = z_1 = 12$

State Recovery

Registers: $i_0 = 0, w_0 = 1, j_0 = 7, k_0 = 0, z_0 = 0$:



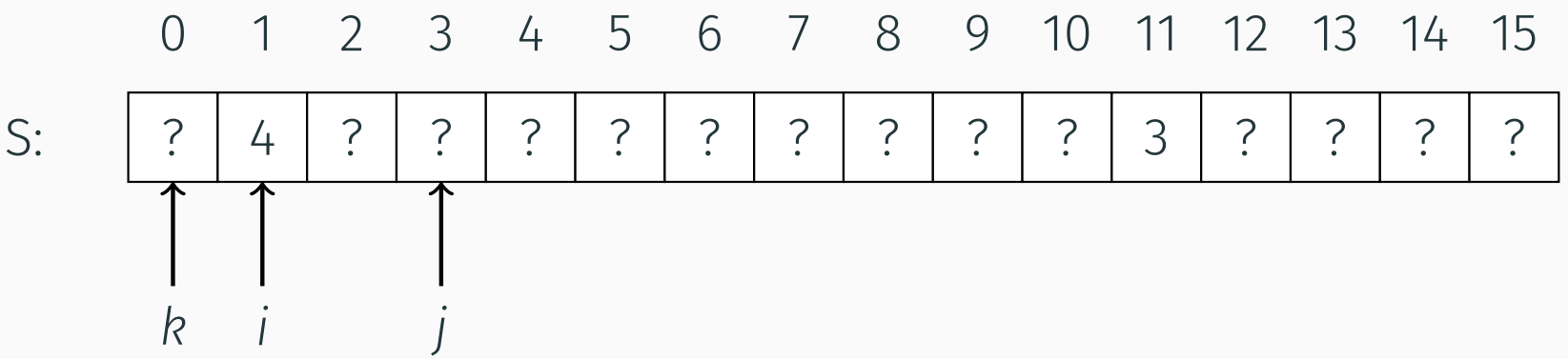

```
Update:
    i = i + w
    j = k + S[j + S[i]]
    k = i + k + S[j]
    SWAP(S[i], S[j])
    z = S[j + S[i + S[z + k]]]
    return z
```

Simulate Update:

- Guess $S[i]$ and $S[j + S[i]]$
- Guess $S[j]$
- Guess $S[z + k]$ and $S[i + S[z + k]]$
- $S[j + S[i + S[z + k]]] = z_1 = 12$

State Recovery

Registers: $i_0 = 0, w_0 = 1, j_0 = 7, k_0 = 0, z_0 = 0$:



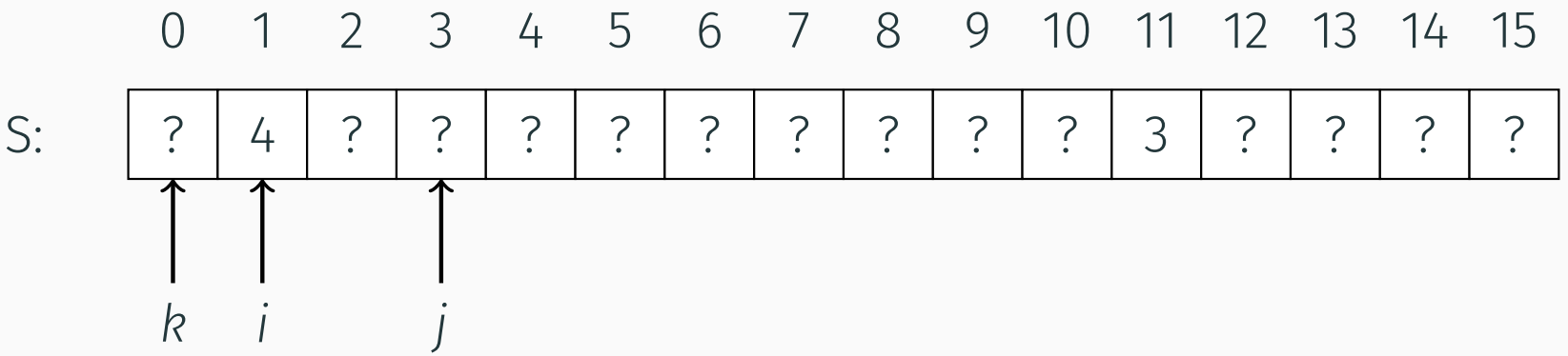

```
Update:
    i = i + w
    j = k + S[j + S[i]]
    k = i + k + S[j]
    SWAP(S[i], S[j])
    z = S[j + S[i + S[z + k]]]
    return z
```

Simulate Update:

- Guess $S[i]$ and $S[j + S[i]]$
- Guess $S[j]$
- Guess $S[z + k]$ and $S[i + S[z + k]]$
- $S[j + S[i + S[z + k]]] = z_1 = 12$

State Recovery

Registers: $i_0 = 0, w_0 = 1, j_0 = 7, k_0 = 0, z_0 = 0$:



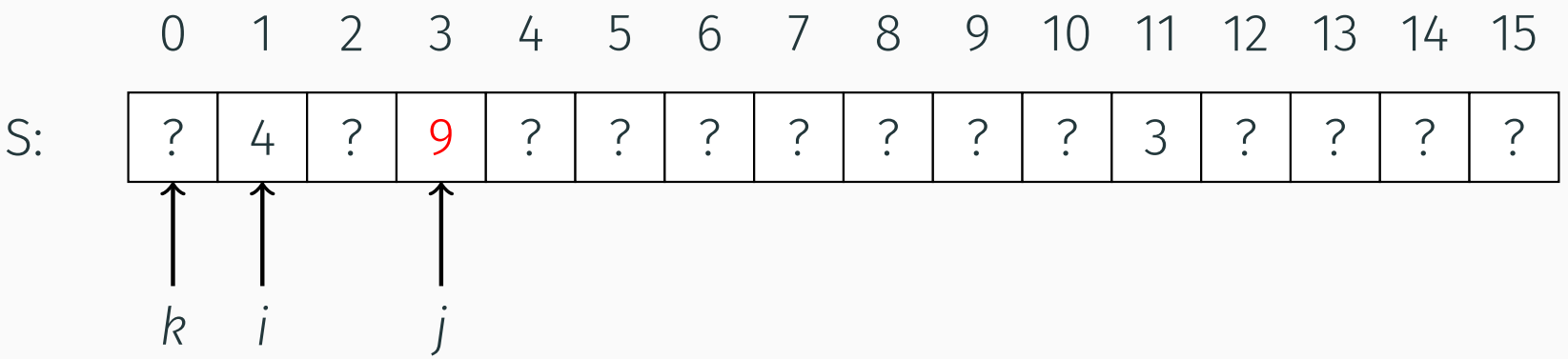

```
Update:
    i = i + w
    j = k + S[j + S[i]]
    k = i + k + S[j]
    SWAP(S[i], S[j])
    z = S[j + S[i + S[z + k]]]
    return z
```

Simulate Update:

- Guess $S[i]$ and $S[j + S[i]]$
- Guess $S[j]$
- Guess $S[z + k]$ and $S[i + S[z + k]]$
- $S[j + S[i + S[z + k]]] = z_1 = 12$

State Recovery

Registers: $i_0 = 0, w_0 = 1, j_0 = 7, k_0 = 0, z_0 = 0$:



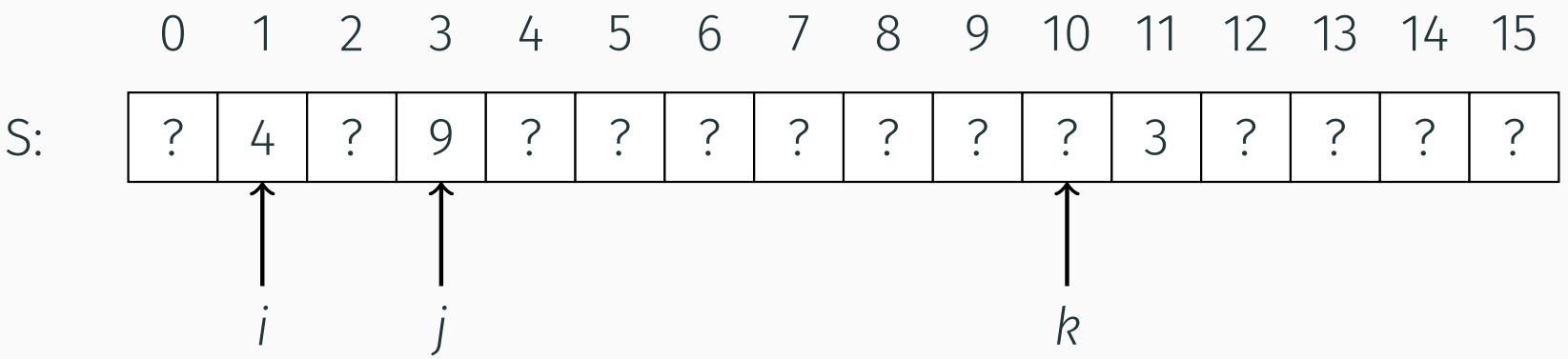

```
Update:
    i = i + w
    j = k + S[j + S[i]]
    k = i + k + S[j]
    SWAP(S[i], S[j])
    z = S[j + S[i + S[z + k]]]
    return z
```

Simulate Update:

- Guess $S[i]$ and $S[j + S[i]]$
- Guess $S[j]$
- Guess $S[z + k]$ and $S[i + S[z + k]]$
- $S[j + S[i + S[z + k]]] = z_1 = 12$

State Recovery

Registers: $i_0 = 0, w_0 = 1, j_0 = 7, k_0 = 0, z_0 = 0$:



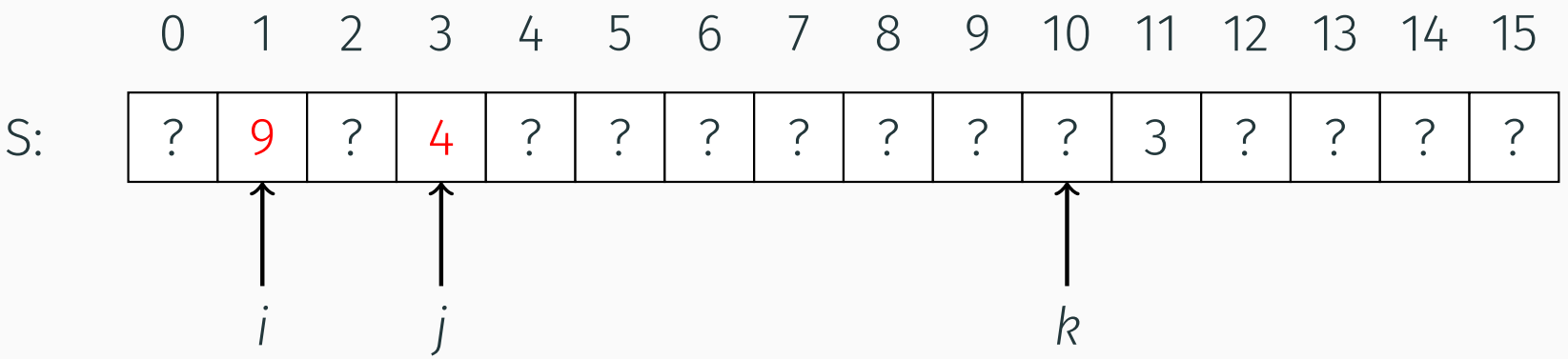

```
Update:
    i = i + w
    j = k + S[j + S[i]]
    k = i + k + S[j]
    SWAP(S[i], S[j])
    z = S[j + S[i + S[z + k]]]
    return z
```

Simulate Update:

- Guess $S[i]$ and $S[j + S[i]]$
- Guess $S[j]$
- Guess $S[z + k]$ and $S[i + S[z + k]]$
- $S[j + S[i + S[z + k]]] = z_1 = 12$

State Recovery

Registers: $i_0 = 0, w_0 = 1, j_0 = 7, k_0 = 0, z_0 = 0$:



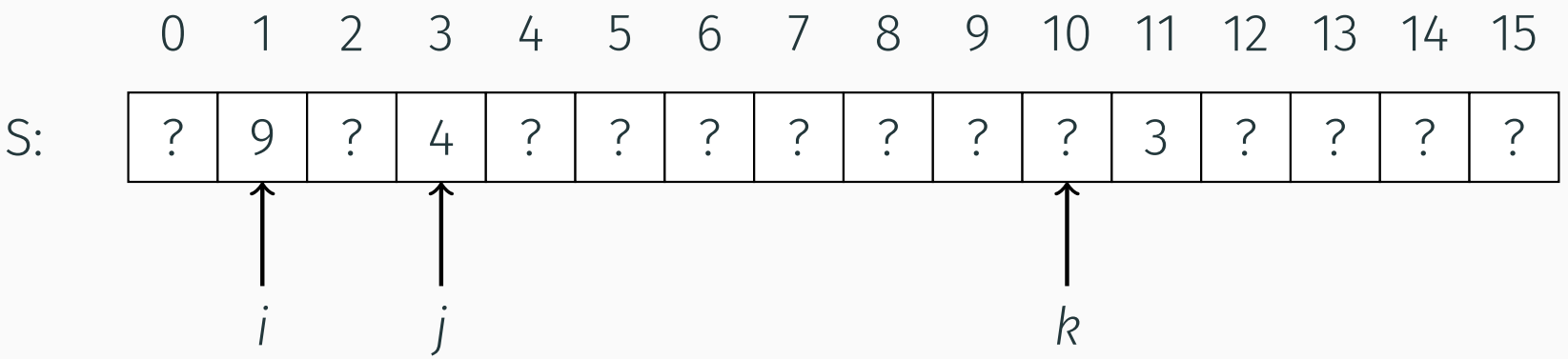

```
Update:
    i = i + w
    j = k + S[j + S[i]]
    k = i + k + S[j]
    SWAP(S[i], S[j])
    z = S[j + S[i + S[z + k]]]
    return z
```

Simulate Update:

- Guess $S[i]$ and $S[j + S[i]]$
- Guess $S[j]$
- Guess $S[z + k]$ and $S[i + S[z + k]]$
- $S[j + S[i + S[z + k]]] = z_1 = 12$

State Recovery

Registers: $i_0 = 0, w_0 = 1, j_0 = 7, k_0 = 0, z_0 = 0$:



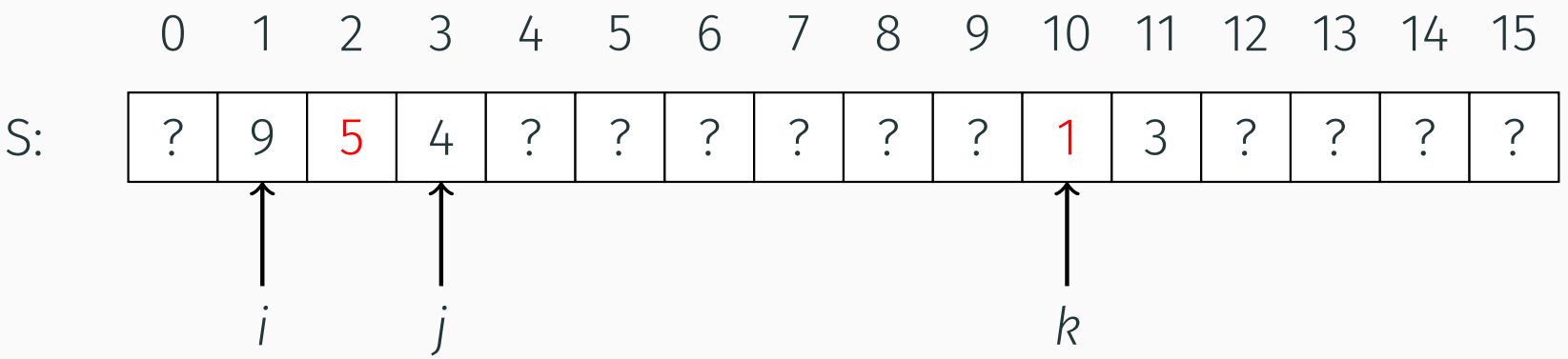

```
Update:
    i = i + w
    j = k + S[j + S[i]]
    k = i + k + S[j]
    SWAP(S[i], S[j])
    z = S[j + S[i + S[z + k]]]
    return z
```

Simulate Update:

- Guess $S[i]$ and $S[j + S[i]]$
- Guess $S[j]$
- Guess $S[z + k]$ and $S[i + S[z + k]]$
- $S[j + S[i + S[z + k]]] = z_1 = 12$

State Recovery

Registers: $i_0 = 0, w_0 = 1, j_0 = 7, k_0 = 0, z_0 = 0$:



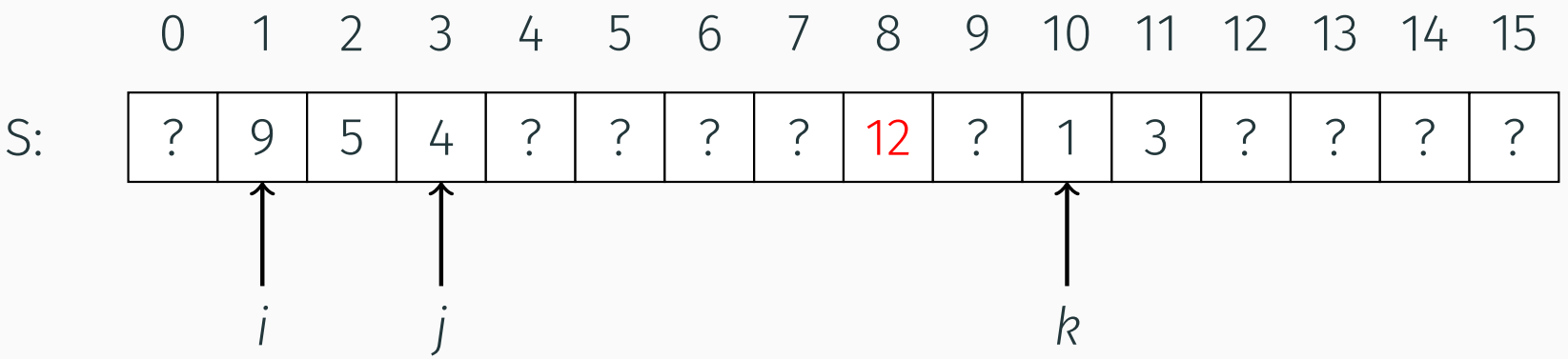

```
Update:
    i = i + w
    j = k + S[j + S[i]]
    k = i + k + S[j]
    SWAP(S[i], S[j])
    z = S[j + S[i + S[z + k]]]
    return z
```

Simulate Update:

- Guess $S[i]$ and $S[j + S[i]]$
- Guess $S[j]$
- Guess $S[z + k]$ and $S[i + S[z + k]]$
- $S[j + S[i + S[z + k]]] = z_1 = 12$

State Recovery

Registers: $i_0 = 0, w_0 = 1, j_0 = 7, k_0 = 0, z_0 = 0$:




```
Update:
    i = i + w
    j = k + S[j + S[i]]
    k = i + k + S[j]
    SWAP(S[i], S[j])
    z = S[j + S[i + S[z + k]]]
    return z
```

Simulate Update:

- Guess $S[i]$ and $S[j + S[i]]$
- Guess $S[j]$
- Guess $S[z + k]$ and $S[i + S[z + k]]$
- $S[j + S[i + S[z + k]]] = z_1 = 12$

State Recovery

Continue until all values of S are known or a contradiction arises:

- An element occurs more than once.
- $z_t \neq S[j_t + S[i_t + S[z_{t-1} + k_t]]]$
- ...

In the paper we compute the complexity of this procedure.

State Recovery

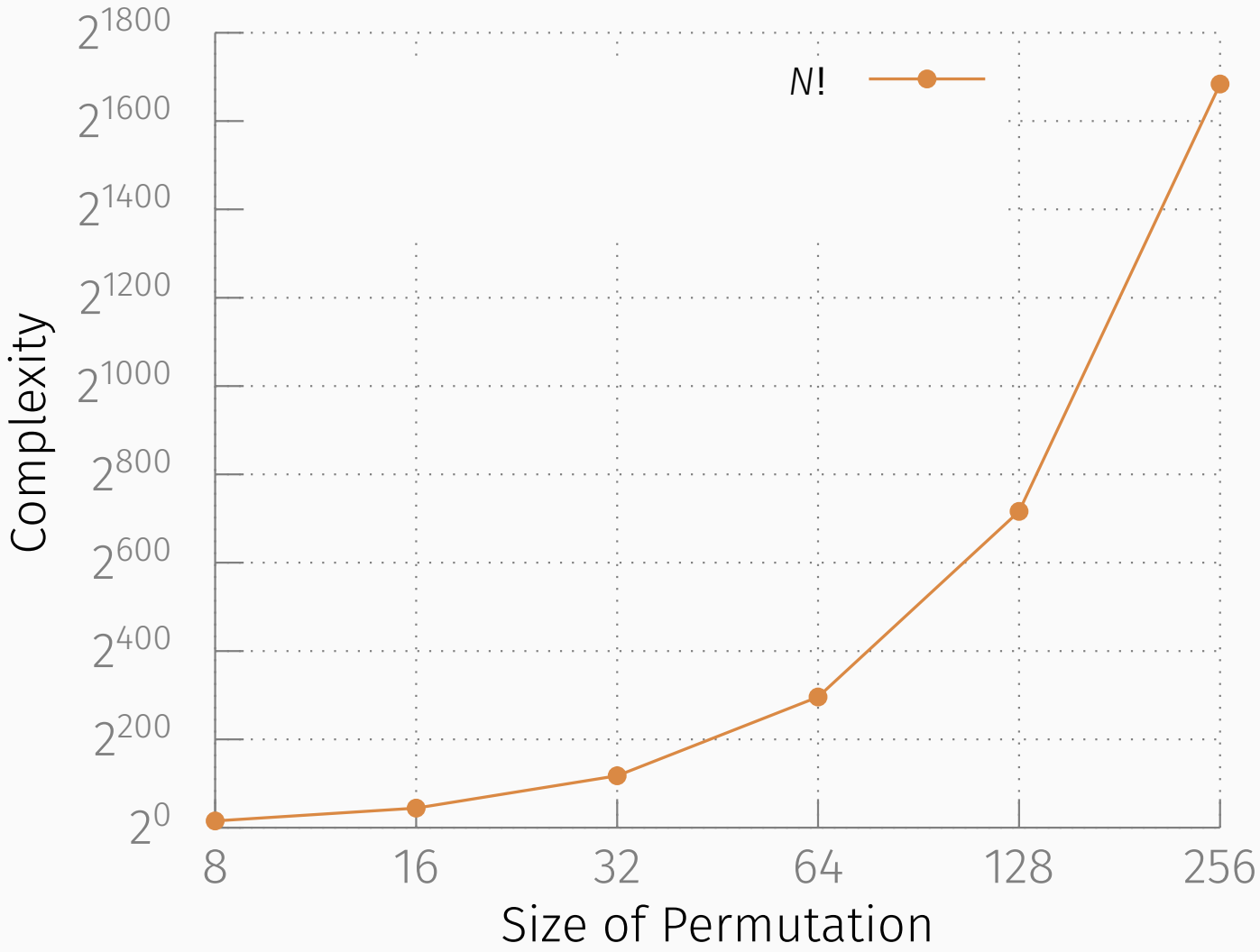
Complexity for different sizes of the permutation table:

N	time comp.	N!
8	$2^{13.7}$	$2^{15.2}$
16	$2^{44.3}$	$2^{44.2}$
32	$2^{99.8}$	$2^{117.6}$
64	$2^{249.0}$	2^{296}
128	$2^{599.4}$	$2^{716.1}$
256	2^{1400}	$2^{1683.9}$

Data complexity is very low: $\mathcal{O}(N)$

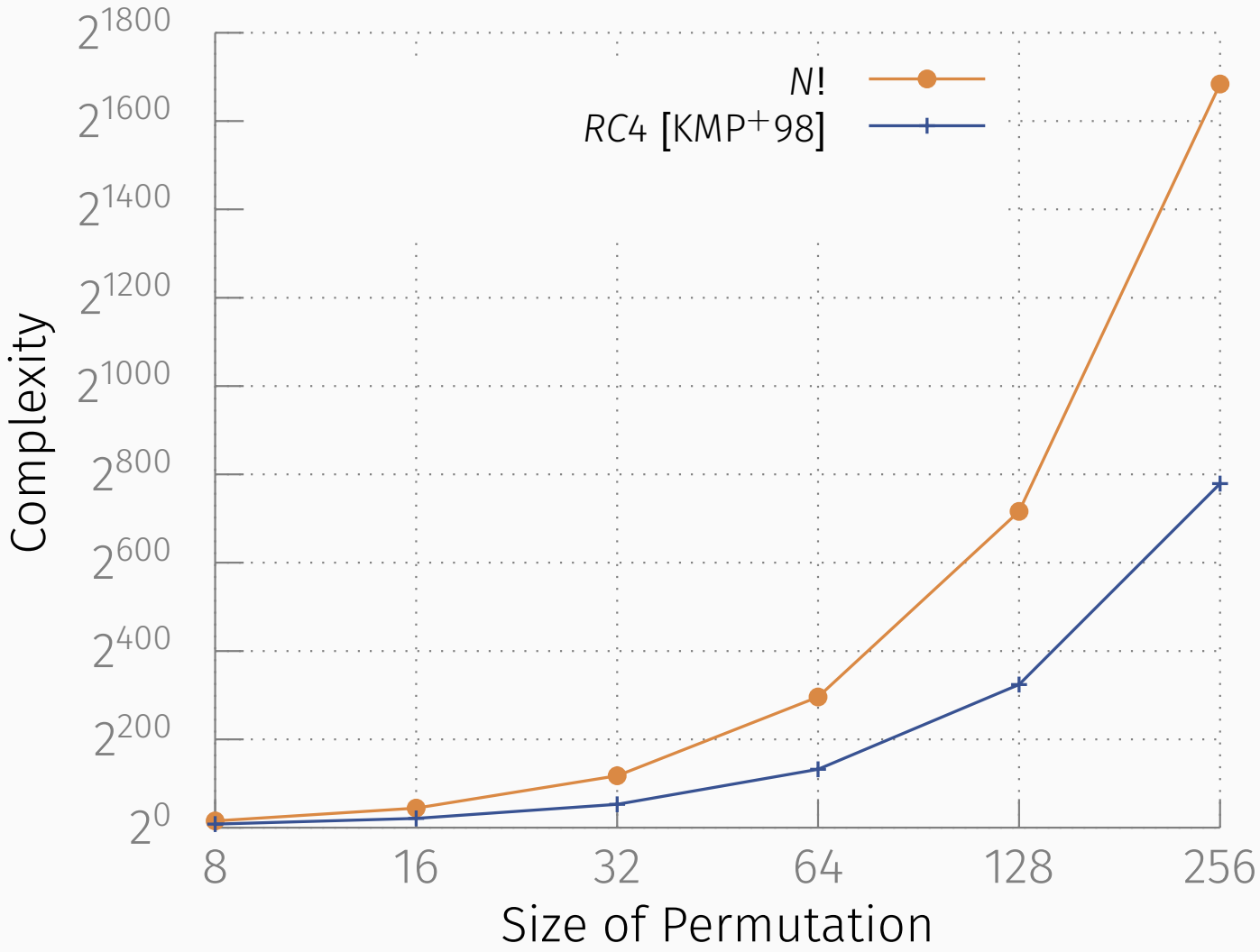
State Recovery

Comparision RC4 and Spritz



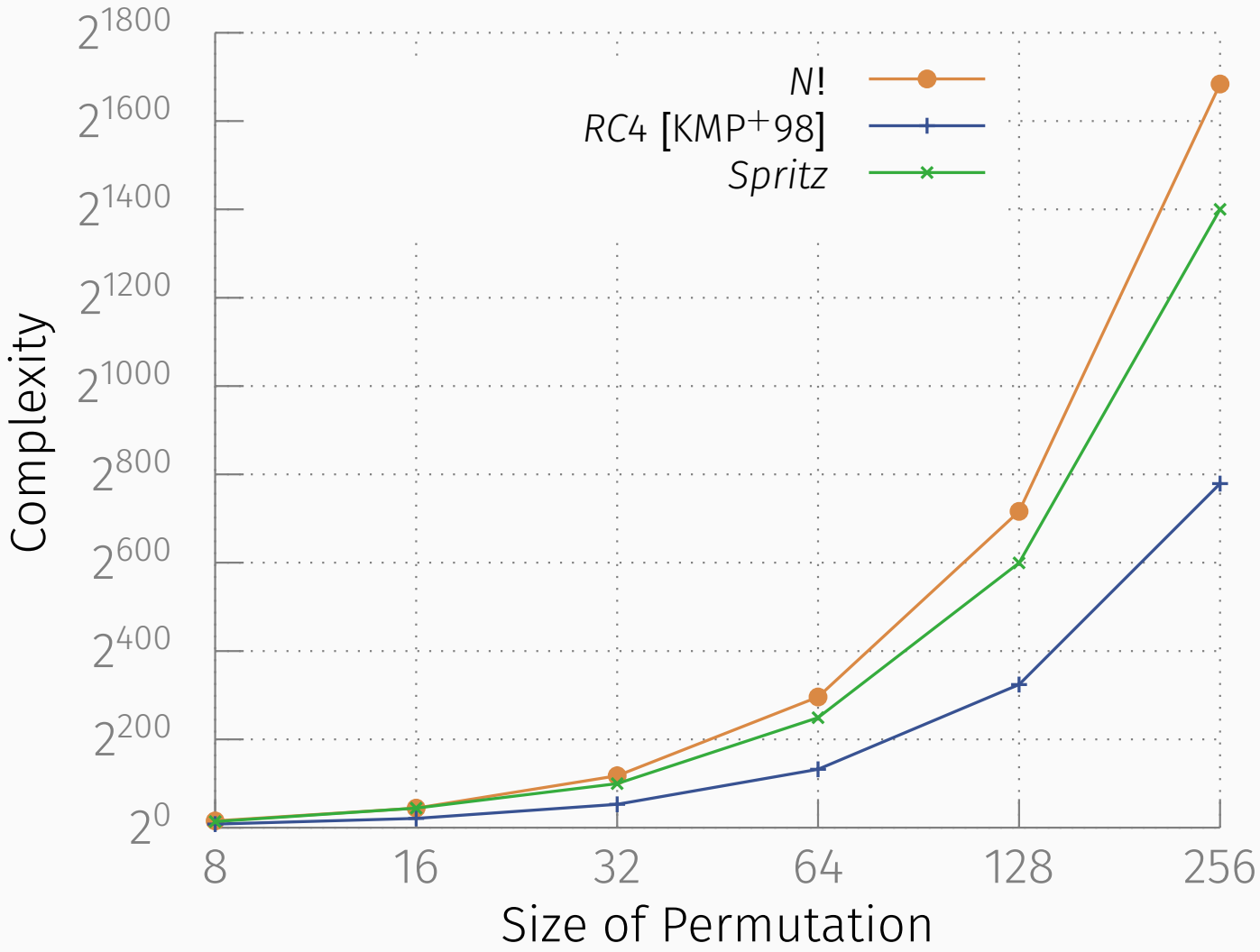
State Recovery

Comparision RC4 and Spritz



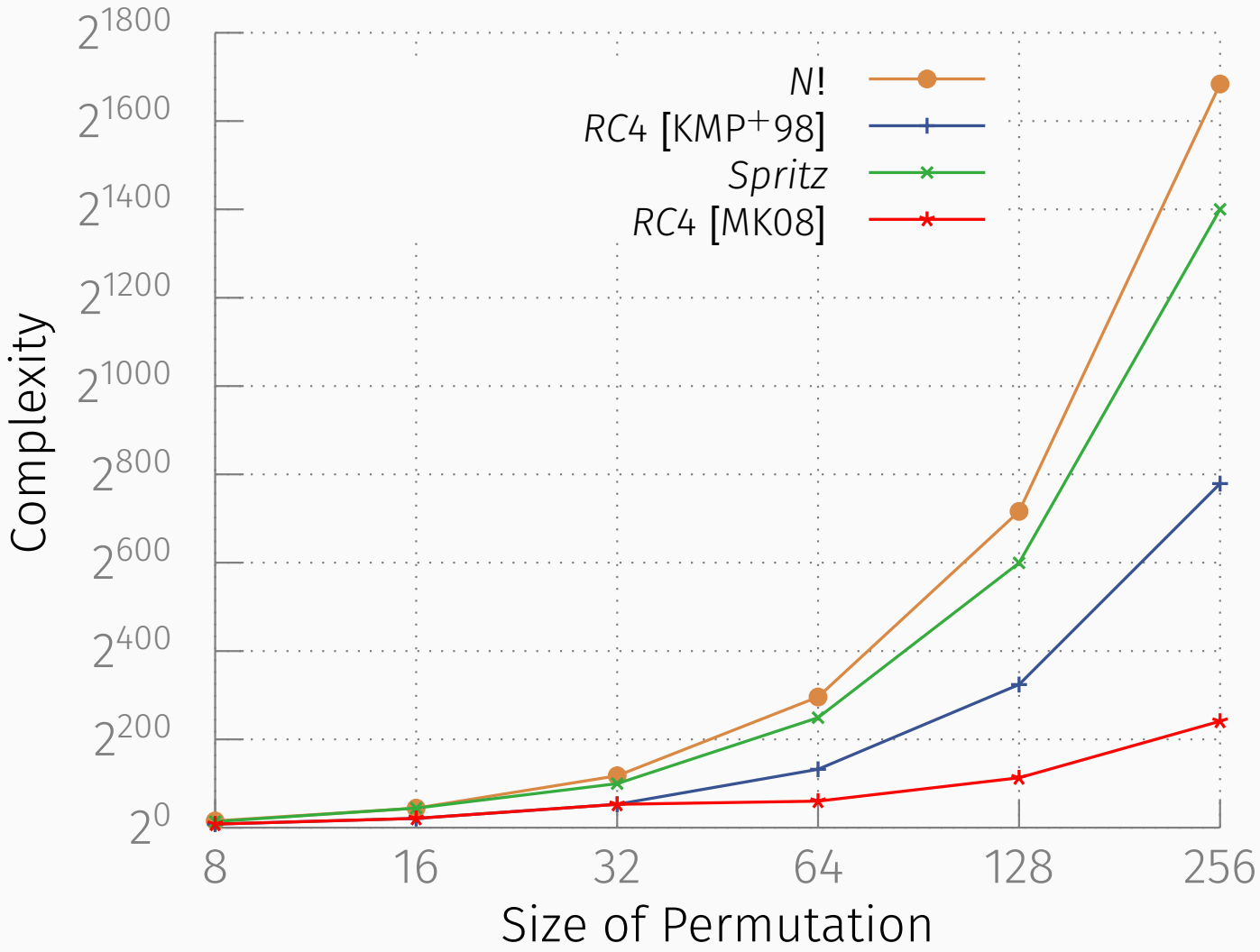
State Recovery

Comparision RC4 and Spritz



State Recovery

Comparision RC4 and Spritz



State Recovery

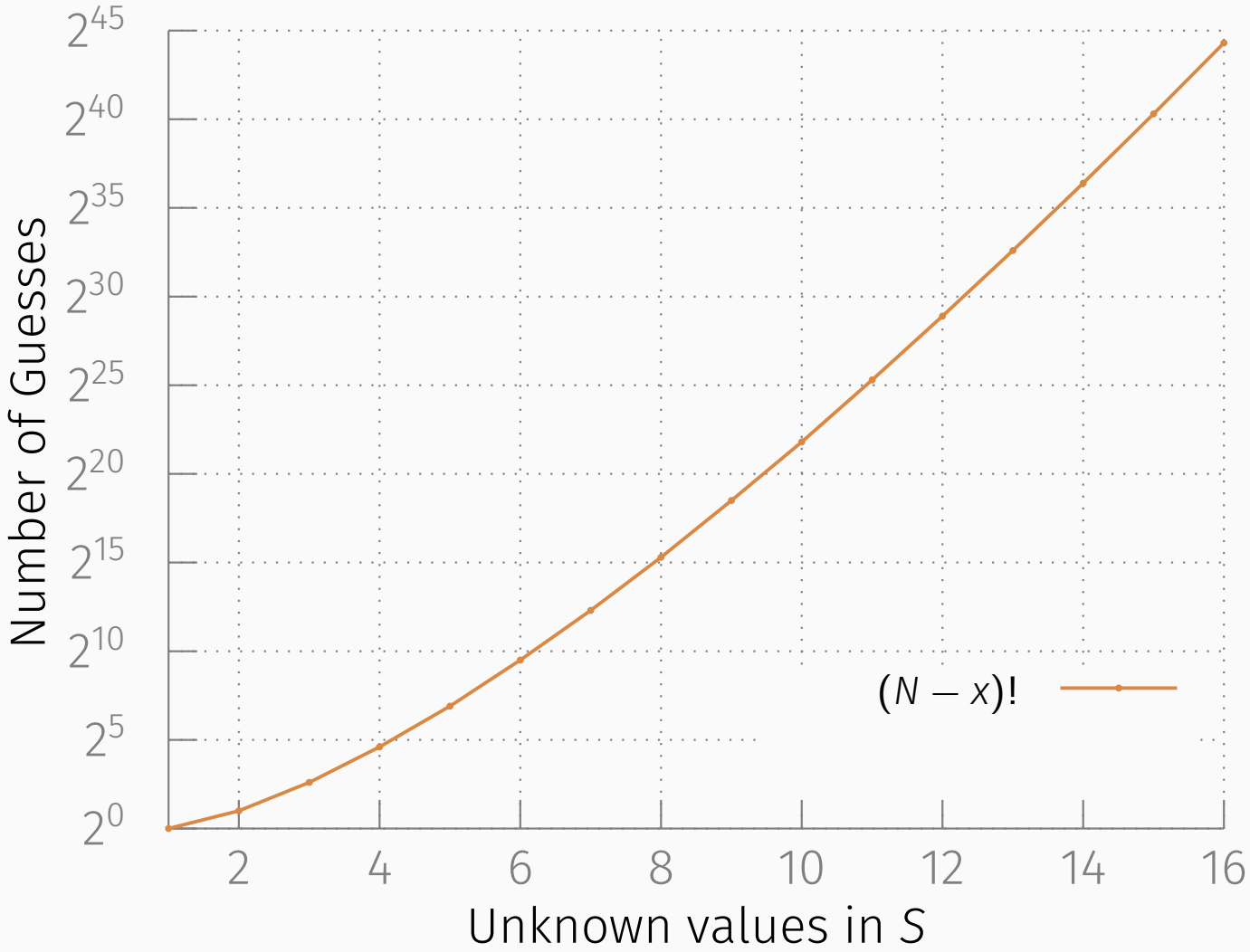
Estimate complexity through experiments

- We also implemented the attack[4].
- Count number of guesses until full state is recovered.
- Pre-assign values to reduce complexity for the experiments.

[4] https://github.com/ralphankele/Spritz

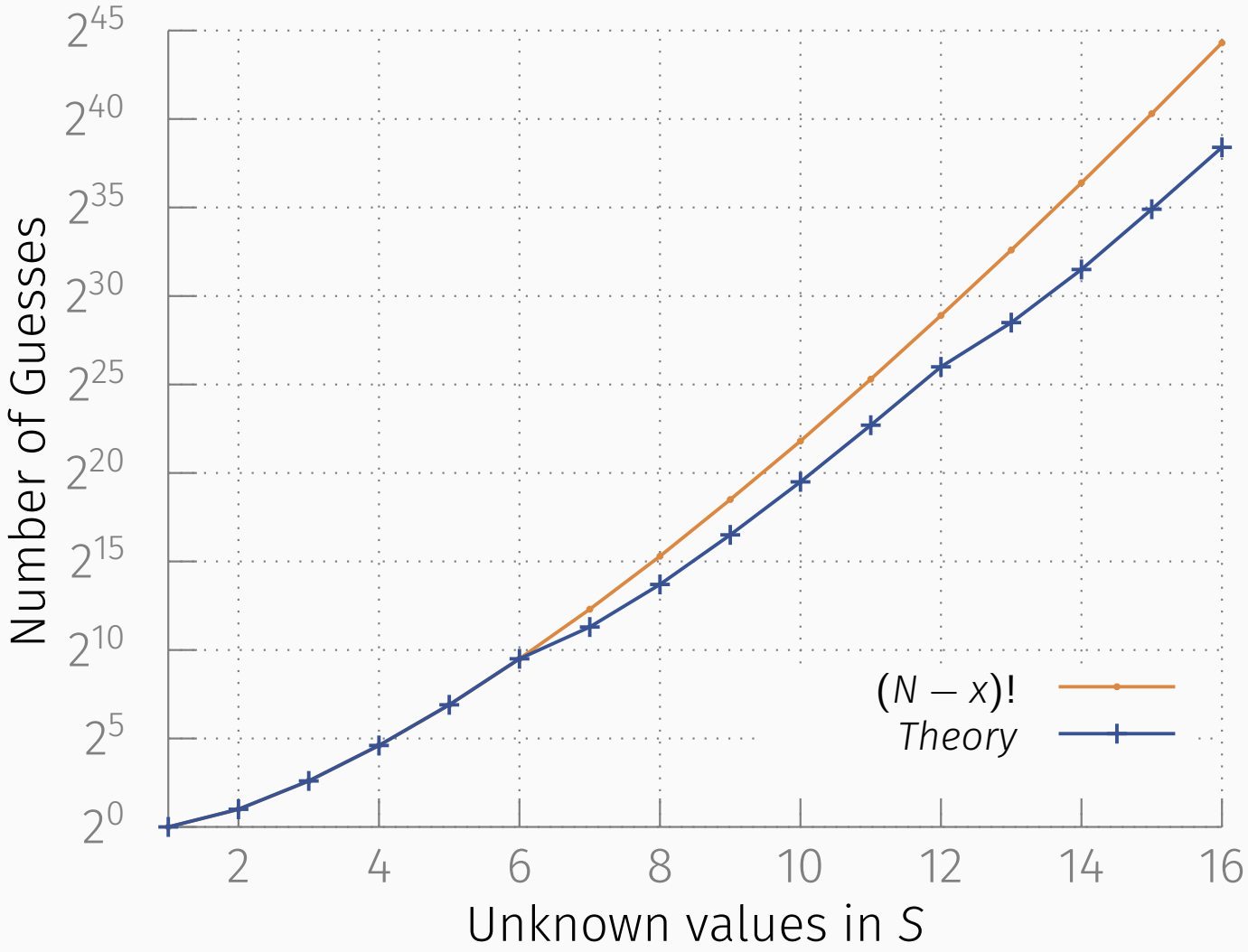
State Recovery

Experimental results for $|S| = 32$:



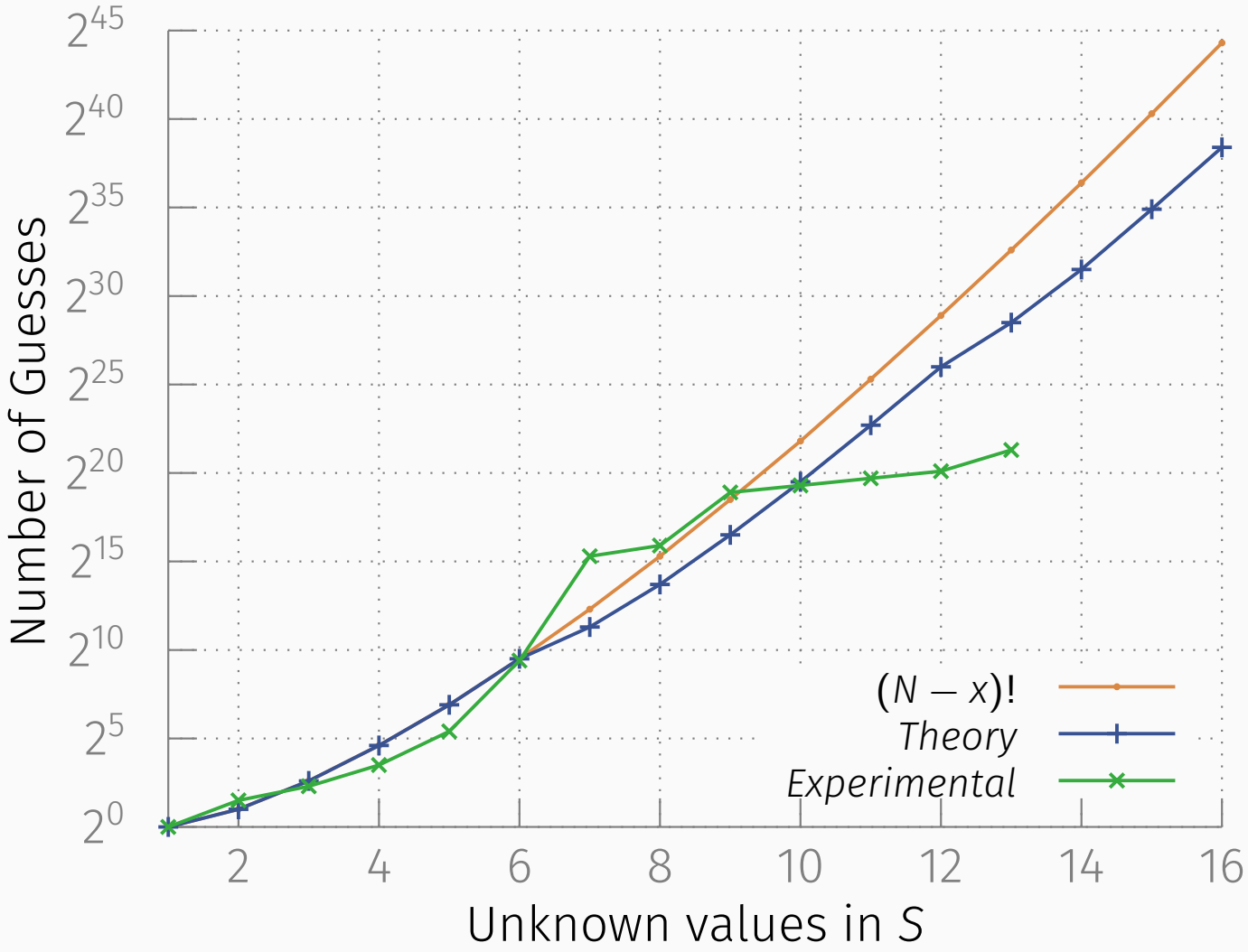
State Recovery

Experimental results for $|S| = 32$:



State Recovery

Experimental results for $|S| = 32$:



Conclusion

Contributions:

- First insights on state recovery attacks on Spritz.
- Estimates for the complexity are very high.
- State-recovery more efficient in experiments.

Possible Improvements:

- Use better search heuristics.
- Combine different methods.
- Utilize longer output stream.

QUESTIONS?

References I

- Nadhem J. AlFardan, Daniel J. Bernstein, Kenneth G. Paterson, Bertram Poettering, and Jacob C. N. Schuldt, *On the security of RC4 in TLS*, Proceedings of the 22th USENIX Security Symposium, Washington, DC, USA, August 14-16, 2013, 2013, pp. 305–320.
- Garman Christina, Kenneth G. Paterson, and Van der Merwe Thyla, *Attacks only get better: Password recovery attacks against RC4 in TLS*, 25th USENIX Security Symposium, 2015, pp. 113–200.
- Scott R. Fluhrer, Itsik Mantin, and Adi Shamir, *Weaknesses in the key scheduling algorithm of RC4*, Selected Areas in Cryptography, SAC 2001, 2001.

References II

- Sourav Sen Gupta, Subhamoy Maitra, Goutam Paul, and Santanu Sarkar, *Proof of empirical RC4 biases and new key correlations*, Selected Areas in Cryptography, SAC 2011, 2011.
- Lars R. Knudsen, Willi Meier, Bart Preneel, Vincent Rijmen, and Sven Verdoolaege, *Analysis methods for (alleged) RC4*, Advances in Cryptology - ASIACRYPT '98, vol. 1514, 1998.
- Vanhoef Mathy and Piessens Frank, *All your biases belong to us: Breaking RC4 in WPA-TKIP and TLS*, 25th USENIX Security Symposium, 2015, pp. 97–112.
- Ilya Mironov, *(not so) random shuffles of RC4*, Advances in Cryptology - CRYPTO 2002, 2002.

References III

 Alexander Maximov and Dmitry Khovratovich, *New state recovery attack on RC4*, Advances in Cryptology – CRYPTO 2008, 2008.