Radboud University

A New Leakage Exploitation Framework and Its Application to Authenticated Encryption

Vahid Jahandideh, Léo Weissbart, Bart Mennink, Lejla Batina
Radboud University (The Netherlands)

NIST Lightweight Cryptography Workshop
June 21-22, 2023

Side-channel analysis (SCA) aware cipher design.

- Ciphers in the real world should be protected against SCA.

Motivation

Side-channel analysis (SCA) aware cipher design.

- Ciphers in the real world should be protected against SCA.
- But, protections are usually costly.

Motivation

Side-channel analysis (SCA) aware cipher design.

- Ciphers in the real world should be protected against SCA.
- But, protections are usually costly.
- So, ciphers are desired to help to lower this cost.

Side-channel analysis (SCA) aware cipher design.

- Ciphers in the real world should be protected against SCA.
- But, protections are usually costly.
- So, ciphers are desired to help to lower this cost.
- Yet, there are no fully clear design guidelines.

Motivation

Side-channel analysis (SCA) aware cipher design.

- Ciphers in the real world should be protected against SCA.
- But, protections are usually costly.
- So, ciphers are desired to help to lower this cost.
- Yet, there are no fully clear design guidelines.

We contribute in this direction by:

Motivation

Side-channel analysis (SCA) aware cipher design.

- Ciphers in the real world should be protected against SCA.
- But, protections are usually costly.
- So, ciphers are desired to help to lower this cost.
- Yet, there are no fully clear design guidelines.

We contribute in this direction by:

- Building a new differential power analysis (DPA) framework.

Motivation

Side-channel analysis (SCA) aware cipher design.

- Ciphers in the real world should be protected against SCA.
- But, protections are usually costly.
- So, ciphers are desired to help to lower this cost.
- Yet, there are no fully clear design guidelines.

We contribute in this direction by:

- Building a new differential power analysis (DPA) framework.
- Applying this framework to NIST LWC (2018-2023) finalists.

Motivation

Side-channel analysis (SCA) aware cipher design.

- Ciphers in the real world should be protected against SCA.
- But, protections are usually costly.
- So, ciphers are desired to help to lower this cost.
- Yet, there are no fully clear design guidelines.

We contribute in this direction by:

- Building a new differential power analysis (DPA) framework.
- Applying this framework to NIST LWC (2018-2023) finalists.
- Showing how mode and primitive design impact leakage mitigation.

Our approach

Starting:

- We perform DPA on 32-bit software XOR instructions.
- Demonstrate how execution leakage can uncover its operands.
- This experiment forms the basis of our framework.

Our approach

Starting:

- We perform DPA on 32-bit software XOR instructions.
- Demonstrate how execution leakage can uncover its operands.
- This experiment forms the basis of our framework.

Applying:

- To recover a secret, we need to find enough XOR operations.

Our approach

Starting:

- We perform DPA on 32-bit software XOR instructions.
- Demonstrate how execution leakage can uncover its operands.
- This experiment forms the basis of our framework.

Applying:

- To recover a secret, we need to find enough XOR operations.

Illustrating:

- We demonstrate the applicability of the approach on ASCON.

Our approach

Starting:

- We perform DPA on 32-bit software XOR instructions.
- Demonstrate how execution leakage can uncover its operands.
- This experiment forms the basis of our framework.

Applying:

- To recover a secret, we need to find enough XOR operations.

Illustrating:

- We demonstrate the applicability of the approach on ASCON.

Deploying:

- Using this framework, we study the SC properties of the finalists.

DPA in a nutshell

Assumption. $p_i \in_r \{0,1\}^w$ are known and $k \in \{0,1\}^w$ is unknown. For N tries, indexed by i, $f(p_i \oplus k)$ is computed, and the power trace is recorded as $\mathbf{T}_i$.

Assumption. $p_i \in_r \{0,1\}^w$ are known and $k \in \{0,1\}^w$ is unknown. For N tries, indexed by i, $f(p_i \oplus k)$ is computed, and the power trace is recorded as $\mathbf{T}_i$.

Main SC observation. If $\mathbf{T}_i$ and $f(p_i \oplus k)$ are dependent, and f is "sufficiently" non-linear, then with enough $\{p_i, \mathbf{T}_i\}_{i=1}^{N}$, an attacker can recover k.

DPA in a nutshell

Assumption. $p_i \in_r \{0,1\}^w$ are known and $k \in \{0,1\}^w$ is unknown. For N tries, indexed by i, $f(p_i \oplus k)$ is computed, and the power trace is recorded as $\mathbf{T}_i$.

Main SC observation. If $\mathbf{T}_i$ and $f(p_i \oplus k)$ are dependent, and f is "sufficiently" non-linear, then with enough $\{p_i, \mathbf{T}_i\}_{i=1}^{N}$, an attacker can recover k.

In practice,

- If $y_i = f(p_i \oplus k)$ is an intermediate variable, $\mathbf{T}_i$ has the required dependency.

DPA in a nutshell

Assumption. $p_i \in_r \{0,1\}^w$ are known and $k \in \{0,1\}^w$ is unknown. For N tries, indexed by i, $f(p_i \oplus k)$ is computed, and the power trace is recorded as $\mathbf{T}_i$.

Main SC observation. If $\mathbf{T}_i$ and $f(p_i \oplus k)$ are dependent, and f is "sufficiently" non-linear, then with enough $\{p_i, \mathbf{T}_i\}_{i=1}^N$, an attacker can recover k.

In practice,

- If $y_i = f(p_i \oplus k)$ is an intermediate variable, $\mathbf{T}_i$ has the required dependency.
- Usually, f is an S-box function.

DPA in a nutshell

Assumption. $p_i \in_r \{0,1\}^w$ are known and $k \in \{0,1\}^w$ is unknown. For N tries, indexed by i, $f(p_i \oplus k)$ is computed, and the power trace is recorded as $\mathbf{T}_i$.

Main SC observation. If $\mathbf{T}_i$ and $f(p_i \oplus k)$ are dependent, and f is "sufficiently" non-linear, then with enough $\{p_i, \mathbf{T}_i\}_{i=1}^N$, an attacker can recover k.

In practice,

- If $y_i = f(p_i \oplus k)$ is an intermediate variable, $\mathbf{T}_i$ has the required dependency.
- Usually, f is an S-box function.
- w is bit-width of the implementation, usually $w \in \{8, 32, 64\}$.

DPA in a nutshell

Assumption. $p_i \in_r \{0,1\}^w$ are known and $k \in \{0,1\}^w$ is unknown. For N tries, indexed by i, $f(p_i \oplus k)$ is computed, and the power trace is recorded as $\mathbf{T}_i$.

Main SC observation. If $\mathbf{T}_i$ and $f(p_i \oplus k)$ are dependent, and f is "sufficiently" non-linear, then with enough $\{p_i, \mathbf{T}_i\}_{i=1}^N$, an attacker can recover k.

In practice,

- If $y_i = f(p_i \oplus k)$ is an intermediate variable, $\mathbf{T}_i$ has the required dependency.
- Usually, f is an S-box function.
- w is bit-width of the implementation, usually $w \in \{8, 32, 64\}$.
- Chunks of the target secret are processed similarly through f. This divide-and-conquer property is required for DPA attacks.

DPA challenges

Limitations. For some finalists, the requirements of DPA are not satisfied.

DPA challenges

Limitations. For some finalists, the requirements of DPA are not satisfied. For example,

- In ASCON: for tag generation, the key is XORed with the state directly (f is identity).

DPA challenges

Limitations. For some finalists, the requirements of DPA are not satisfied. For example,

- In ASCON: for tag generation, the key is XORed with the state directly (f is identity).
- In ISAP: inside the initialization, divide-and-conquer is thwarted.

DPA challenges

Limitations. For some finalists, the requirements of DPA are not satisfied. For example,

- In ASCON: for tag generation, the key is XORed with the state directly (f is identity).
- In ISAP: inside the initialization, divide-and-conquer is thwarted.
- In Xoodyak: during encryption, if nonce uniqueness is preserved, N is bounded.

DPA challenges

Limitations. For some finalists, the requirements of DPA are not satisfied. For example,

- In ASCON: for tag generation, the key is XORed with the state directly (f is identity).
- In ISAP: inside the initialization, divide-and-conquer is thwarted.
- In Xoodyak: during encryption, if nonce uniqueness is preserved, N is bounded.

Relaxing the requirements. What happens if f is linear or identity function? Can we still recover k given enough $\{p_i, \mathbf{T}_i\}_{i=1}^{N}$?

Experimenting with XOR

Assembly code snippet to test XOR leakage.

```
...        : r3 = k   r9 = p
8000aa8: bf00           nop
8000aaa: ea89 0903      eor.w  r9, r9, r3 ; r9 = r9 ⊕ r3
8000aae: bf00           nop
```

Heading/trailing nops to filter out effects of neighboring operations.

Experimenting with XOR

Assembly code snippet to test XOR leakage.

```
...        : r3 = k  r9 = p
8000aa8: bf00          nop
8000aaa: ea89 0903     eor.w  r9, r9, r3 ; r9 = r9 ⊕ r3
8000aae: bf00          nop
```

Heading/trailing nops to filter out effects of neighboring operations.

Setup:

- A chipwhisperer CW308 UFO board using ARMv7-M architecture.
- Trace collection is synchronized at 7.37 MHz.

Experimenting with XOR

Assembly code snippet to test XOR leakage.

```
...         : r3 = k  r9 = p
8000aa8: bf00        nop
8000aaa: ea89 0903   eor.w  r9, r9, r3 ; r9 = r9 ⊕ r3
8000aae: bf00        nop
```

Heading/trailing nops to filter out effects of neighboring operations.

Setup:

- A chipwhisperer CW308 UFO board using ARMv7-M architecture.
- Trace collection is synchronized at 7.37 MHz.

Processing tools:

- Correlation power analysis (CPA), linear regression (LR), combined with deep learning (DL) techniques.

Results:

- Attacks work with around 10K traces using 100K for profiling.

More on the attack tools

Leakage model. For some noise n and constant a, the leakage buried in the traces is described as $l = a\mathsf{HW}(k \oplus p) + n$.

More on the attack tools

Leakage model. For some noise n and constant a, the leakage buried in the traces is described as $l = a\text{HW}(k \oplus p) + n$.

Byte by Byte CPA. To estimate byte j of the key, the attacker computes empirical correlation $\text{cor}(\text{HW}(k^* \oplus p[j]), \mathbf{T})$ for all values $0 \leq k^* \leq 255$, and sorts the key hypothesis based on the results.

Leakage model. For some noise n and constant a, the leakage buried in the traces is described as $l = a\text{HW}(k \oplus p) + n$.

Byte by Byte CPA. To estimate byte j of the key, the attacker computes empirical correlation $\text{cor}(\text{HW}(k^* \oplus p[j]), \mathbf{T})$ for all values $0 \leq k^* \leq 255$, and sorts the key hypothesis based on the results.

Drawback. In targeting each byte, the value of the other bytes is considered as noise. This increases the number of required traces for a successful attack.

More on the attack tools

Using LR for adjusting weights. Using the traces, leakage l of a variable v in terms of its bits can be estimated with LR as $l = \sum_{j=1}^{w} a_j v[j] + n$.

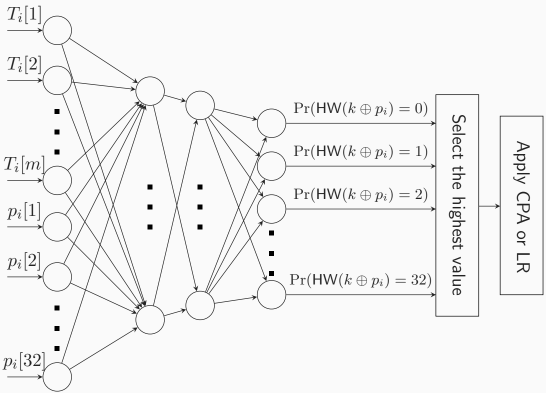
Using LR for adjusting weights. Using the traces, leakage l of a variable v in terms of its bits can be estimated with LR as $l = \sum_{j=1}^{w} a_j v[j] + n$.

Using LR for the attack. If v is $k \oplus p$, output leakage of the XOR can be expressed in basis of bits of p as:

$$l = \sum_{j=1}^{w} a_j (k[j] \oplus p[j]) + n = \sum_{k[j]=0} a_j p[j] + \sum_{k[j]=1} a_j (1 - p[j]) + n$$
$$= \sum_{k[j]=0} a_j p[j] - \sum_{k[j]=1} a_j p[j] + b,$$

for some key-independent value b. The sign of coefficient of $p[j]$ reveals $k[j]$.

Combining CPA/LR with DL. To decrease the number of required traces, HW$(k \oplus p_i)$ can be estimated from $\{p_i, \mathbf{T}_i\}$ with DL. j in $T_i[j]$ and $p_i[j]$ denotes samples and bits, respectively.



Attack Results

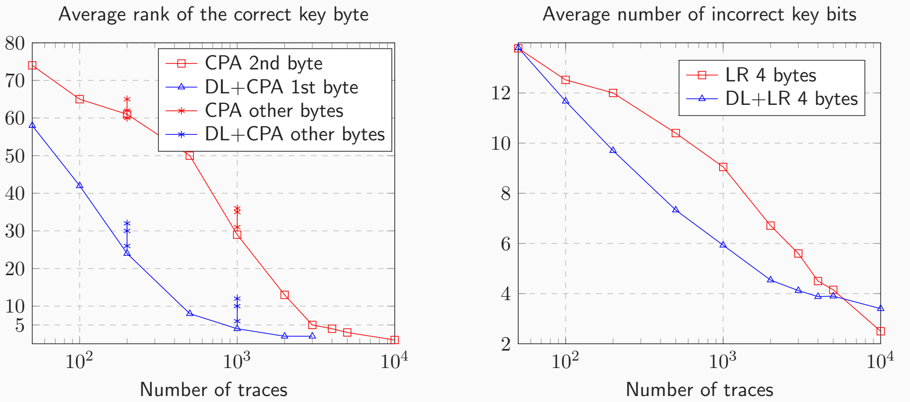


Figure: (Left) CPA with and without DL. The CPA attack for the 2nd byte produces slightly better results. DL+CPA in the used model worked better for the 1st byte. Results for other bytes are also shown. (Right) LR with and without DL.

Assumption. If w-bit chunks of an n-bit target secret are processed in $\frac{n}{w}$ separate XOR operations, each with a known random operand, the attacker can learn that secret.

Our framework

Assumption. If w-bit chunks of an n-bit target secret are processed in $\frac{n}{w}$ separate XOR operations, each with a known random operand, the attacker can learn that secret.

Application. We investigate the running assembly to find enough XOR instructions that are merging random (and known) operands with secret parameters.

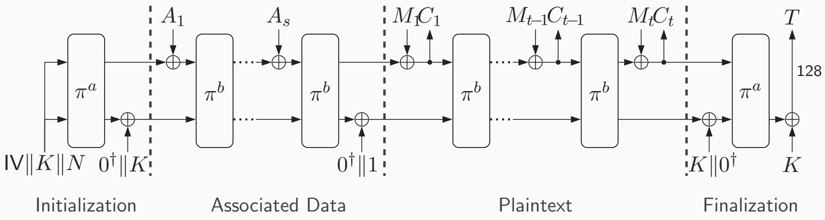


Figure: ASCON-Encryption. a and b denote the number of permutation rounds.

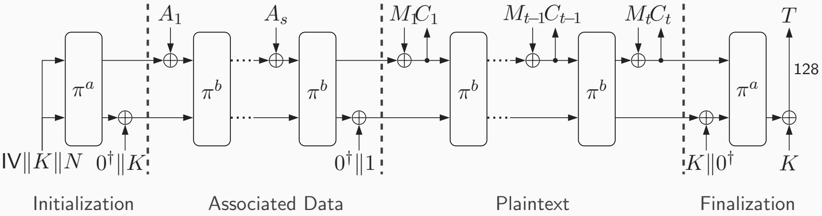
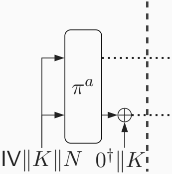


Figure: ASCON-Encryption. a and b denote the number of permutation rounds.

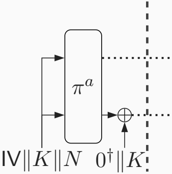
- K is used both in the initialization and finalization phases.
- State recovery in middle phases will not lead to key recovery or tag forgery.
- So, only the initialization and finalization need protection.
- This property is referred to as leveled masking.

Applying our framework. An optimized implementation of the initialization contains 4 32-bit XOR operations. .



Initialization

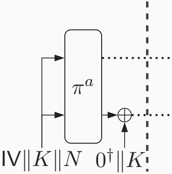
Applying our framework. An optimized implementation of the initialization contains 4 32-bit XOR operations. Instructions for $K1$ and $K3$ are given.



Initialization

```
... ; r0 contains address of parameters
10b60: e9d0 1205  ldrd r1, r2, [r0, #20]; r1 = K2, r2 = N1
10b64: e9d0 7601  ldrd r7, r6, [r0, #4] ; r7 = IV0, r6 = K1
10b68: e9d0 5403  ldrd r5, r4, [r0, #12]; r5 = K0, r4 = K3
... ; r1, r2, r4, r5, r6, r7 are not touched
10b7e: e9d0 ec08  ldrd lr, ip, [r0, #32]; lr = N3, ip = N2
10b82: 69c3       ldr  r3, [r0, #28] ; r3 = N0
10b84: f8d0 8000  ldr.w r8, [r0] ; r8 = IV1
10b88: f084 04f0  eor.w r4, r4, #240 ; K3 = K3 ⊕ 0xf0
10b8c: ea86 0904  eor.w r9, r6, r4 ; r9 = K1 ⊕ K3
10b90: ea88 0a0e  eor.w sl, r8, lr ; sl = IV1 ⊕ N3
10b94: ea82 0b0e  eor.w fp, r2, lr ; fp = N1 ⊕ N3
10b98: ea62 0e0e  orn  lr, r2, lr ; lr = N1|(~ N3)
10b9c: ea8e 0e09  eor.w lr, lr, r9; lr = r9 ⊕ lr = K1 ⊕ K3 ⊕ N1|(~ N3)
10ba0: ea82 0206  eor.w r2, r2, r6 ; r2 = N1 ⊕ K1
```

Applying our framework. An optimized implementation of the initialization contains 4 32-bit XOR operations. Instructions for $K1$ and $K3$ are given.



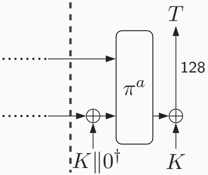
Initialization

```
... ; r0 contains address of parameters
10b60: e9d0 1205  ldrd r1, r2, [r0, #20]; r1 = K2, r2 = N1
10b64: e9d0 7601  ldrd r7, r6, [r0, #4] ; r7 = IV0, r6 = K1
10b68: e9d0 5403  ldrd r5, r4, [r0, #12]; r5 = K0, r4 = K3
... ; r1, r2, r4, r5, r6, r7 are not touched
10b7e: e9d0 ec08  ldrd lr, ip, [r0, #32]; lr = N3, ip = N2
10b82: 69c3       ldr  r3, [r0, #28] ; r3 = N0
10b84: f8d0 8000  ldr.w r8, [r0] ; r8 = IV1
10b88: f084 04f0  eor.w r4, r4, #240 ; K3 = K3 ⊕ 0xf0
10b8c: ea86 0904  eor.w r9, r6, r4 ; r9 = K1 ⊕ K3
10b90: ea88 0a0e  eor.w sl, r8, lr ; sl = IV1 ⊕ N3
10b94: ea82 0b0e  eor.w fp, r2, lr ; fp = N1 ⊕ N3
10b98: ea62 0e0e  orn  lr, r2, lr ; lr = N1|(~ N3)
10b9c: ea8e 0e09  eor.w lr, lr, r9; lr = r9 ⊕ lr = K1 ⊕ K3 ⊕ N1|(~ N3)
10ba0: ea82 0206  eor.w r2, r2, r6 ; r2 = N1 ⊕ K1
```

A profiling phase is required to identify the sample index of target instructions.

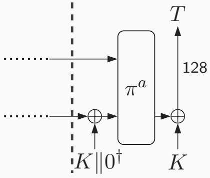
In the finalization phase, for generating tag, words of K are operands of 4 XOR operations. The corresponding instructions for $K0$ are highlighted.

In the finalization phase, for generating tag, words of K are operands of 4 XOR operations. The corresponding instructions for $K0$ are highlighted.



Finalization

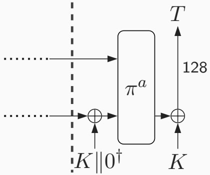
In the finalization phase, for generating tag, words of K are operands of 4 XOR operations. The corresponding instructions for $K0$ are highlighted.



Finalization

... ; `r4` *address of state, r5 address of key*
109ae: e9d5 0100 **ldrd r0**, **r1**, [**r5**] ; $\texttt{r0} = K0, r1 = K1$
109b2: 69a2 **ldr r2**, [**r4**, #24] ; $\texttt{r2} = S0$
109b4: 69e3 **ldr r3**, [**r4**, #28] ; $\texttt{r3} = S1$
109b6: 4050 **eors r0**, **r2** ; $\texttt{r0} = K0 \oplus S0$ $(T0 = r0)$
... ; $\texttt{r0}, \texttt{r2}$ *are not touched and r4 has not changed*

In the finalization phase, for generating tag, words of K are operands of 4 XOR operations. The corresponding instructions for $K0$ are highlighted.



Finalization

```
... ; r4 address of state, r5 address of key
109ae: e9d5 0100  ldrd r0, r1, [r5] ; r0 = K0, r1 = K1
109b2: 69a2       ldr  r2, [r4, #24] ; r2 = S0
109b4: 69e3       ldr  r3, [r4, #28] ; r3 = S1
109b6: 4050       eors r0, r2     ; r0 = K0 ⊕ S0    (T0 = r0)
... ; r0, r2 are not touched and r4 has not changed
```

K is also XORed with the state before the final permutation. However, there is no known data associated with this instruction. These operations can be targeted with a second-order DPA.

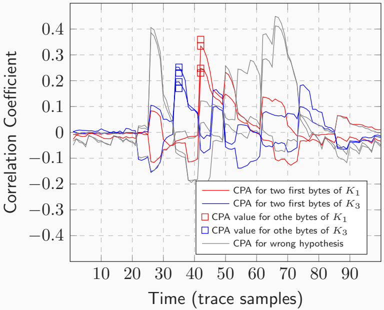


Figure: CPA results for attacking the initialization phase of ASCON.

Application to ASCON

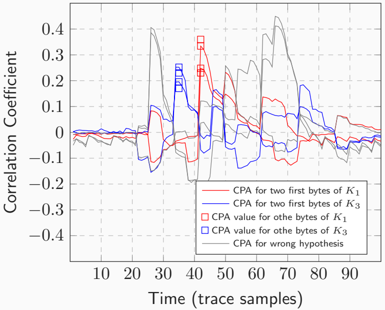


Figure: CPA results for attacking the initialization phase of ASCON.

- The interval for $K1$ is one instruction after that of $K3$.

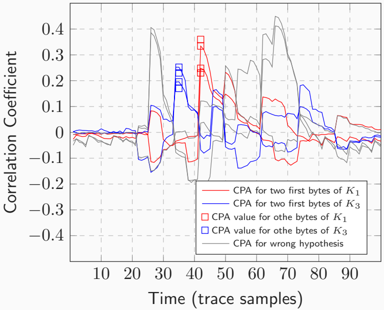


Figure: CPA results for attacking the initialization phase of ASCON.

- The interval for $K1$ is one instruction after that of $K3$.
- Peaks should be compared only in their target interval.

Application to ASCON

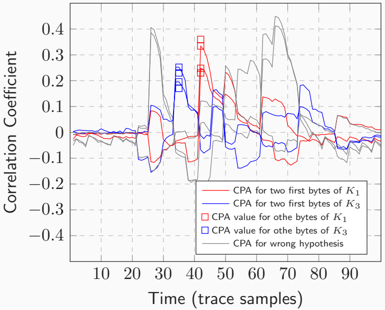


Figure: CPA results for attacking the initialization phase of ASCON.

- The interval for $K1$ is one instruction after that of $K3$.
- Peaks should be compared only in their target interval.
- The attack at the finalization works, but peaks are negative.

DPA for plaintext recovery. To decrypt C_1^* (without knowing a valid tag), an attacker can ask for decryption of random C_1 using the same (N^*, A^*) and perform DPA over the XOR defined by $M_1^* = C_1^* \oplus \text{Trunc}(S^*, r)$.

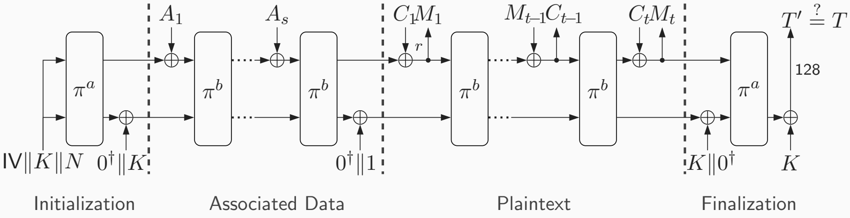


Figure: ASCON-Decryption

DPA for plaintext recovery. To decrypt C_1^* (without knowing a valid tag), an attacker can ask for decryption of random C_1 using the same (N^*, A^*) and perform DPA over the XOR defined by $M_1^* = C_1^* \oplus \text{Trunc}(S^*, r)$.

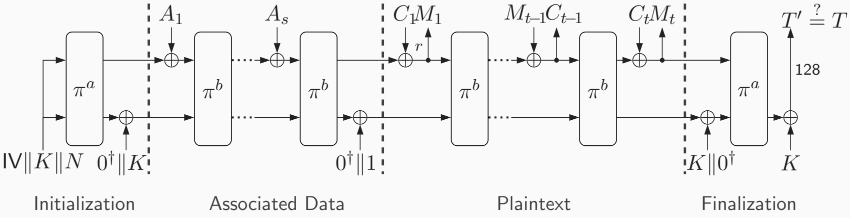


Figure: ASCON-Decryption

Encrypt-then-MAC construction can prevent plaintext recovery attacks.

Application to Xoodyak

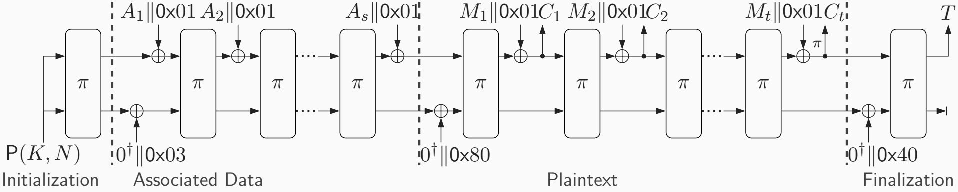


Figure: Xoodyak-Encryption, with P$(K,N) = K\|N\|$0x80$\|$0x01$\|0^\dagger\|$0x02.

Application to Xoodyak

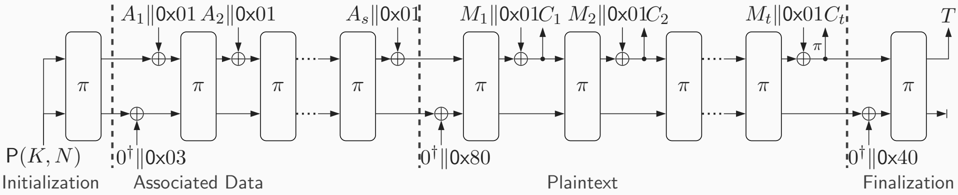


Figure: Xoodyak-Encryption, with P$(K, N) = K||N||$0x80$||$0x01$||0^\dagger||$0x02.

- We skip the initialization and focus on the associated data absorption phase.

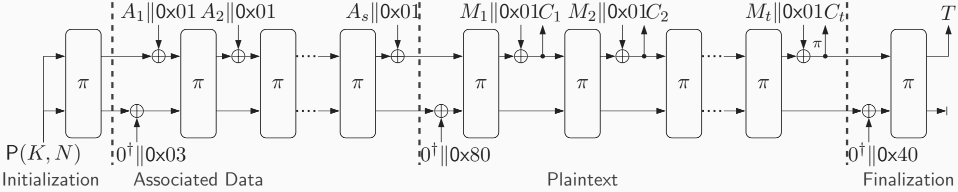


Figure: Xoodyak-Encryption, with P$(K, N) = K||N||$0x80$||$0x01$||0^\dagger||$0x02.

- We skip the initialization and focus on the associated data absorption phase.
- Associated data is absorbed with rate $r = 44 \cdot 8$ and state size is $48 \cdot 8$.

Application to Xoodyak

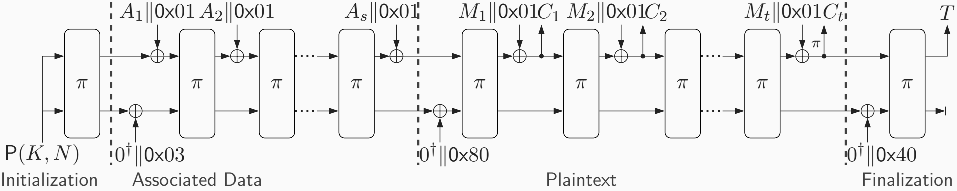


Figure: Xoodyak-Encryption, with $P(K,N) = K||N||0x80||0x01||0^\dagger||0x02$.

- We skip the initialization and focus on the associated data absorption phase.
- Associated data is absorbed with rate $r = 44 \cdot 8$ and state size is $48 \cdot 8$.
- If the attacker is allowed to repeat the nonce, SC attack can recover the state.

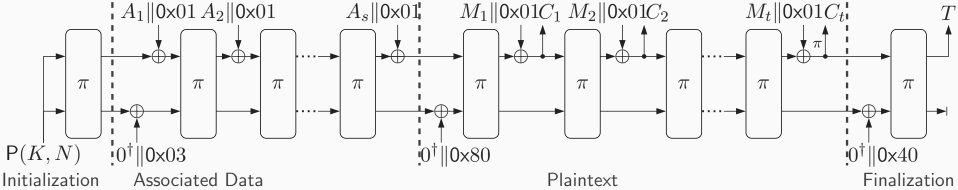


Figure: Xoodyak-Encryption, with $P(K,N) = K\|N\|0x80\|0x01\|0^\dagger\|0x02$.

- We skip the initialization and focus on the associated data absorption phase.
- Associated data is absorbed with rate $r = 44 \cdot 8$ and state size is $48 \cdot 8$.
- If the attacker is allowed to repeat the nonce, SC attack can recover the state.
- The permutation chain upgrades a state recovery attack to a key recovery attack.

Application to the other finalists

Cipher	(A)	(B)	(C)	(D)	(E)	(F)	(G)
ASCON	✓			32 bit	✓	×	
Elephant	✓			8 bit	×	✓	
GIFT-COFB	✓			32 bit	×	×	
Grain128-AEAD	×	×	✓	1 bit	✓	×	
ISAP	×	×	×	32 bit	✓	✓	✓
TinyJambu	✓			32 bit	×	×	
Xoodyak	×	✓		8 bit	×	×	×

- (A) is checked if there is a first-order DPA key recovery attack.
- (B) is checked if there is a first-order DPA key recovery attack in the fixed nonce setting.
- (C) is checked if there is only a second-order DPA key recovery attack.
- (D) shows the bit-width of the studied assembly implementation.
- (E) is checked if leveled masking is possible.
- (F) is checked if DPA for plaintext recovery is not possible.
- (G) is checked if there is an SCA-aware version that is not computationally heavier.

Conclusion

- Our study identified XOR operations as a source for differential power analysis.

Conclusion

- Our study identified XOR operations as a source for differential power analysis.
- Our proposed framework effectively evaluates the SC security of ASCON.

Conclusion

- Our study identified XOR operations as a source for differential power analysis.
- Our proposed framework effectively evaluates the SC security of ASCON.
- This approach has been applied successfully to evaluate the SC security of NIST LWC finalists.

Conclusion

- Our study identified XOR operations as a source for differential power analysis.
- Our proposed framework effectively evaluates the SC security of ASCON.
- This approach has been applied successfully to evaluate the SC security of NIST LWC finalists.
- However, it requires knowledge of the running assembly to carry out the attack.

Conclusion

- Our study identified XOR operations as a source for differential power analysis.
- Our proposed framework effectively evaluates the SC security of ASCON.
- This approach has been applied successfully to evaluate the SC security of NIST LWC finalists.
- However, it requires knowledge of the running assembly to carry out the attack.

Thank you for your attention!