CIPHERH: Automated Detection of Ciphertext Side-channel Vulnerabilities in Cryptographic Implementations

Sen Deng[1], Mengyuan Li[2], Yining Tang[1], Shuai Wang[3], Shoumeng Yan[4], Yinqian Zhang[1]

[1]Southern University of Science and Technology

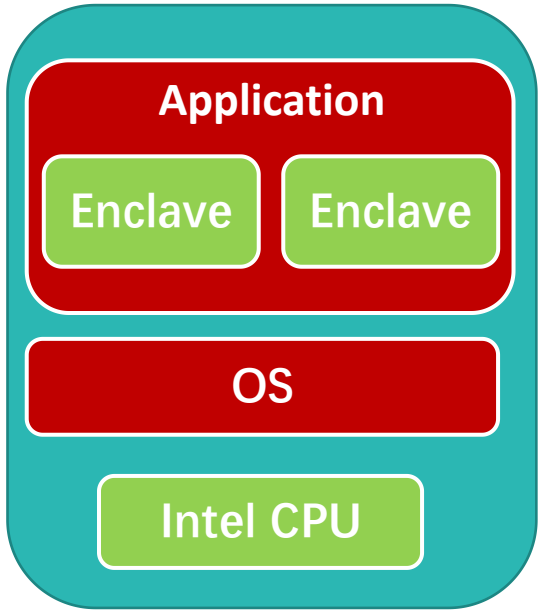
[2]The Ohio State University

[3]Hong Kong University of Science and Technology

[4]The Ant Group

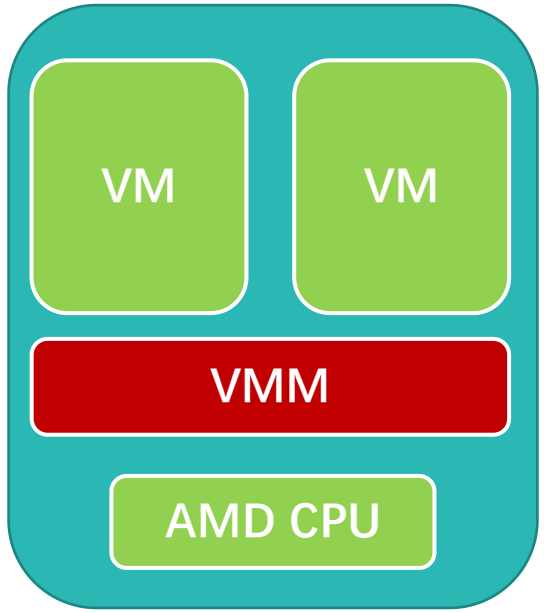
Trusted Execution Environment

Intel SGX



Confidentiality & Integrity

AMD SEV



Microsoft Azure

aws

Google Cloud

Known Attacks

TEE is **not** a *silver bullet*

Hardware Design Attacks

- Unencrypted VMCB
- ASID-based Isolation

Transient Execution Attacks

- Meltdown-like attacks
- Spectre-like attacks

Side Channel Attacks

- Cache-based attacks
- DRAM-based attacks

Memory Corruption Attacks

- Dark-ROP attacks
- Iago attacks

Thread Concurrency Attacks

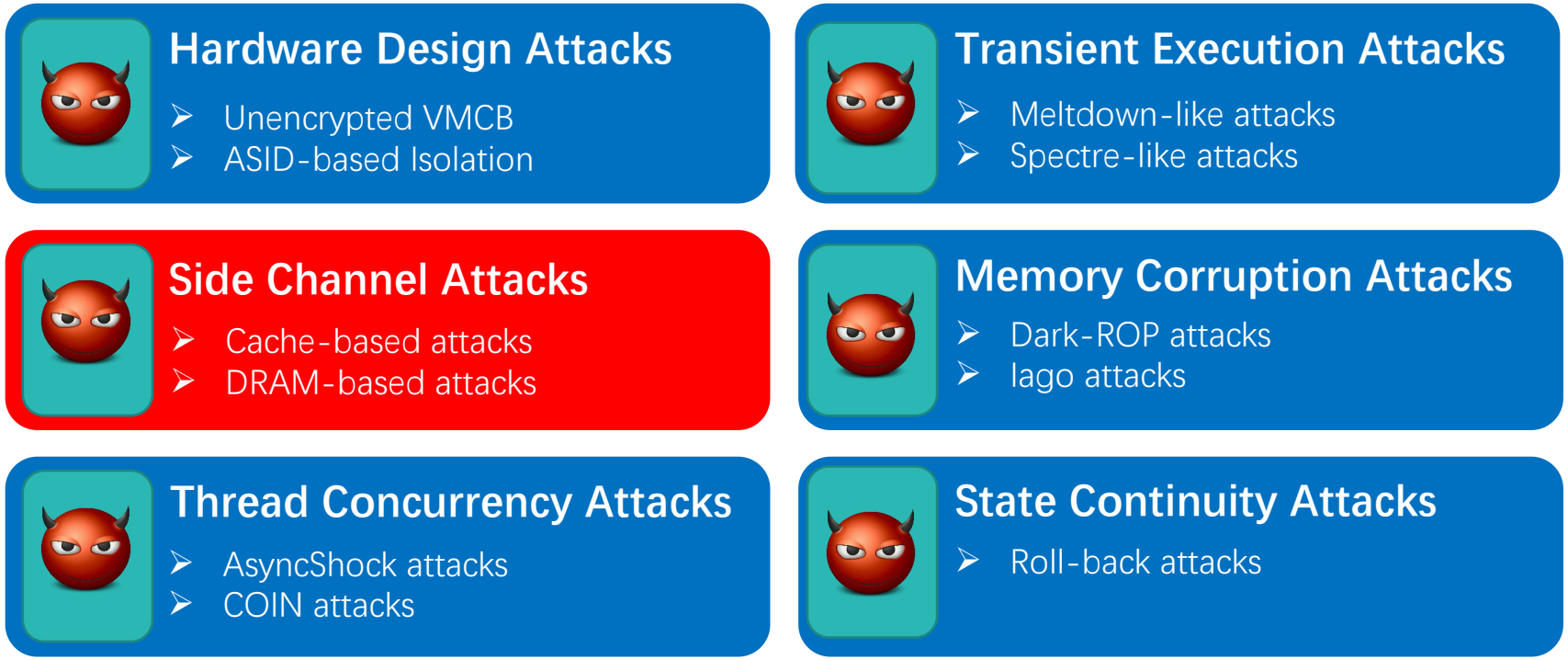
- AsyncShock attacks
- COIN attacks

State Continuity Attacks

- Roll-back attacks

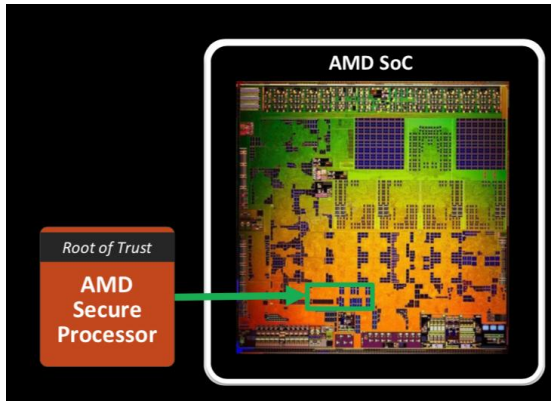
Known Attacks

TEE is **not** a *silver bullet*



Ciphertext Side Channels

A emerging threats and new types of side channels toward TEEs



The ciphertext side-channel was first illustrated in [1] to infer secret register values from the VM Save Area (VMSA) in SEV-SNP.

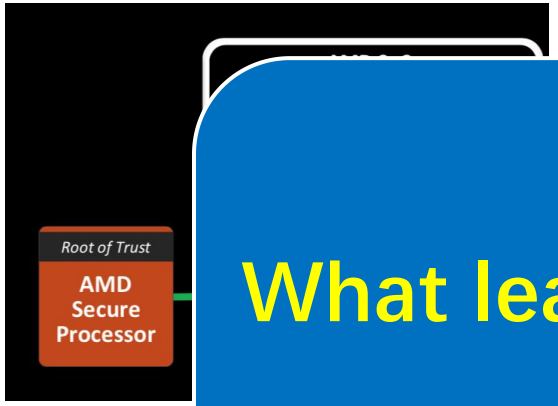
Then it was extended to any memory space including kernel areas, heaps as well as stacks in [2].

[1] Li, Mengyuan, Yinqian Zhang, Huibo Wang, Kang Li, and Yueqiang Cheng. "CIPHERLEAKS: Breaking Constant-time Cryptography on AMD SEV via the Ciphertext Side Channel." In 30th USENIX Security Symposium (USENIX Security 21), pp. 717-732. 2021.
[2]Li, Mengyuan, Luca Wilke, Jan Wichelmann, Thomas Eisenbarth, Radu Teodorescu, and Yinqian Zhang. "A Systematic Look at Ciphertext Side Channels on AMD SEV-SNP." In 2022 IEEE Symposium on Security and Privacy (SP), pp. 1541-1541. IEEE Computer Society, 2022.

For [1], CVE-2020-12966
https://www.amd.com/en/corporate/product-security/bulletin/amd-sb-1013

For [2], CVE-2021-46744
https://www.amd.com/en/corporate/product-security/bulletin/amd-sb-1033

Ciphertext Side Channels



NP.

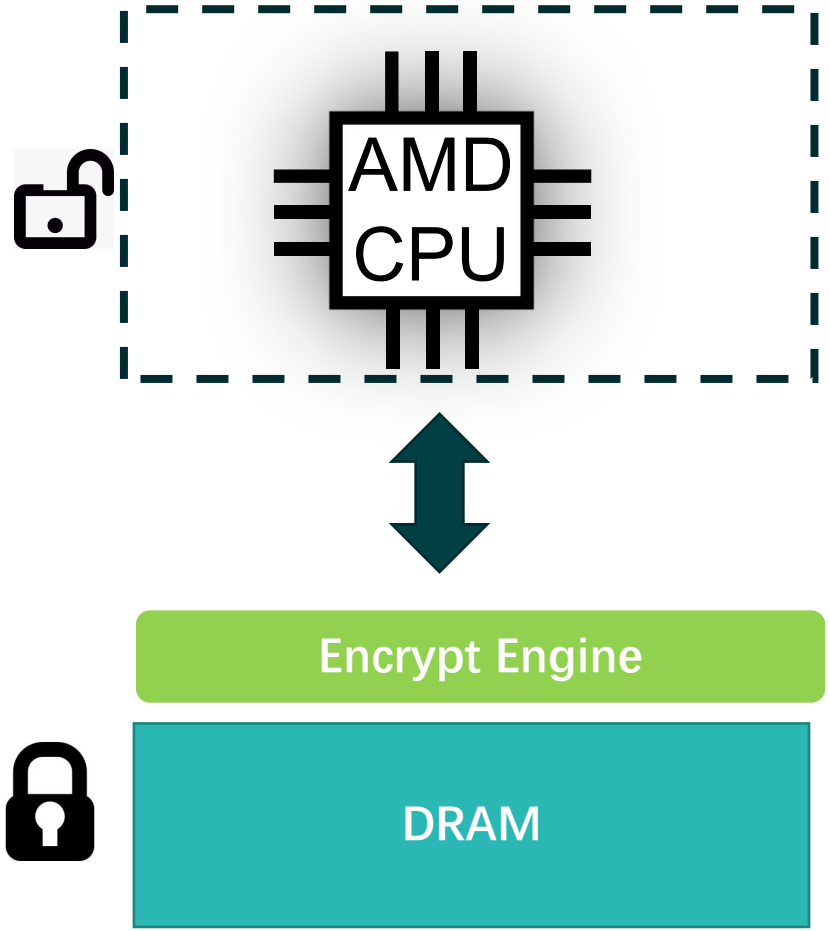
What leads to the Ciphertext Side Channels ?

[1] Li, M phy on AMD SEV
[2]Li, Mengyuan, Luca Wilke, Jan Wichelmann, Thomas Eisenbarth, Radu Teodorescu, and Yinqian Zhang. "A Systematic Look at Ciphertext Side Channels on AMD SEV-SNP." In 2022 IEEE Symposium on Security and Privacy (SP), pp. 1541-1541. IEEE Computer Society, 2022.

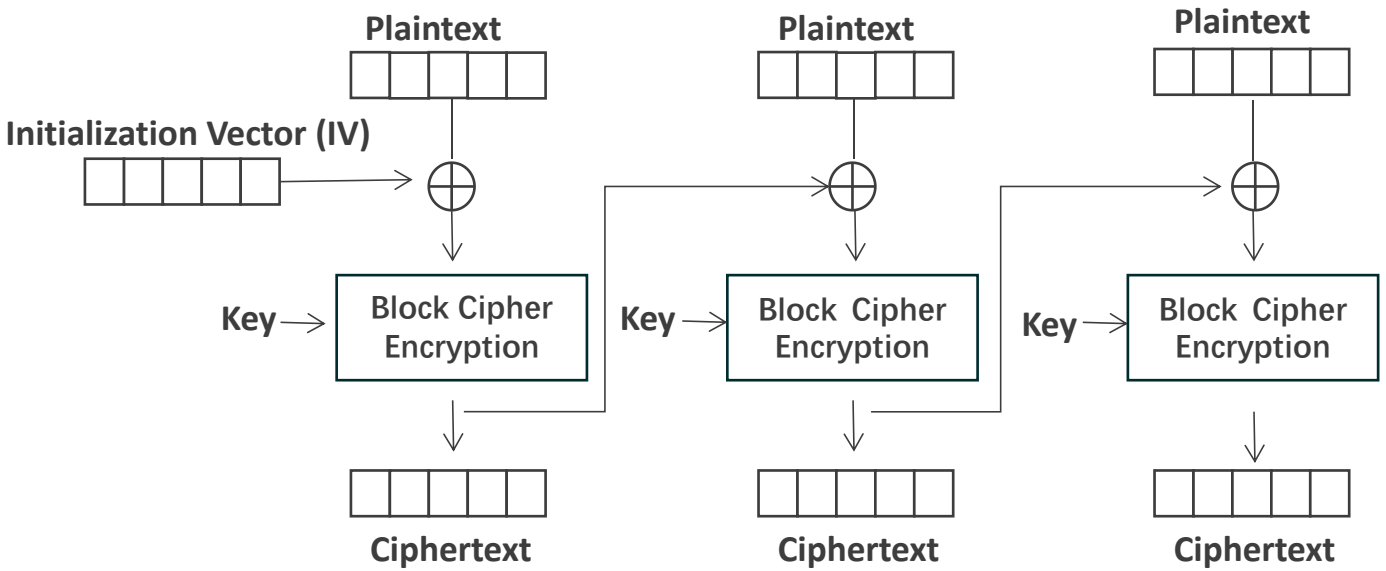
For [1], CVE-2020-12966
https://www.amd.com/en/corporate/product-security/bulletin/amd-sb-1013

For [2], CVE-2021-46744
https://www.amd.com/en/corporate/product-security/bulletin/amd-sb-1033

Hardware Memory Encryption

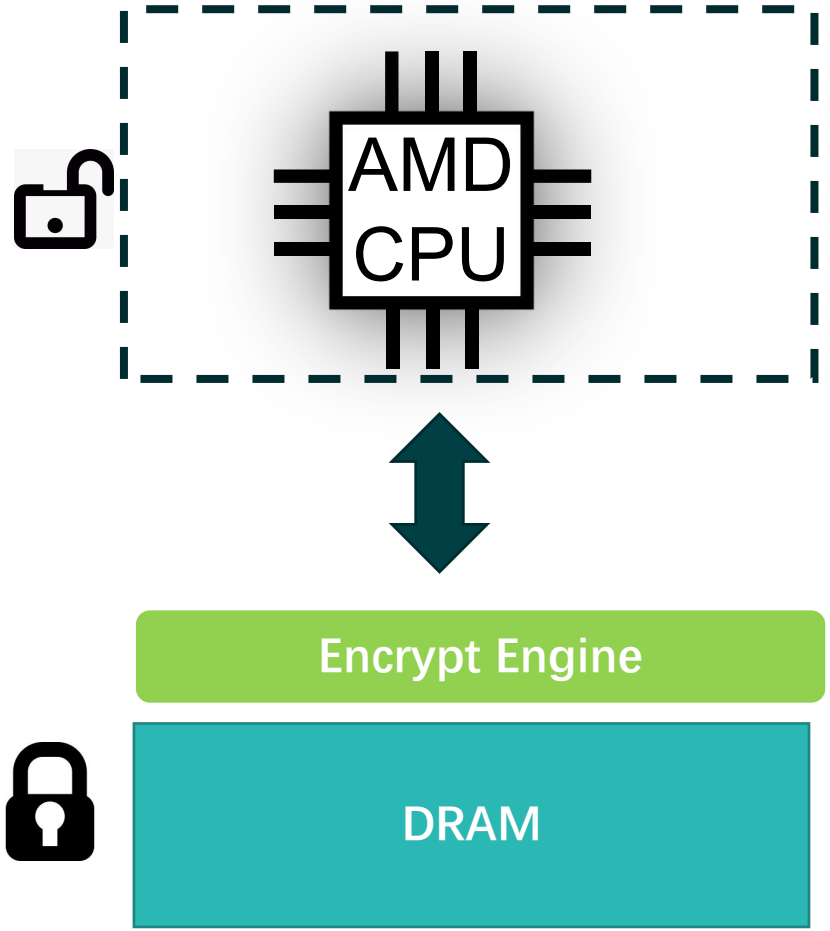


Memory encryption is the primary means to protect memory data against an adversary with either software-level or physical-level access to the memory content.

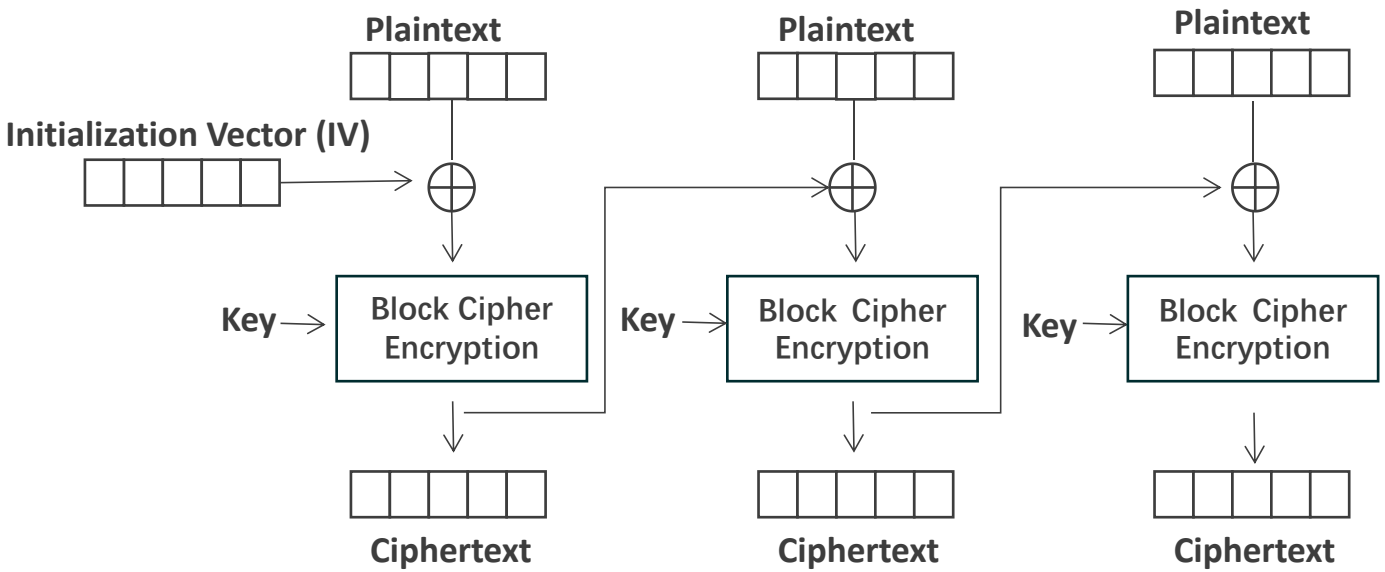


Cipher Block Chaining (CBC) mode encryption

Infeasible Encryption Modes

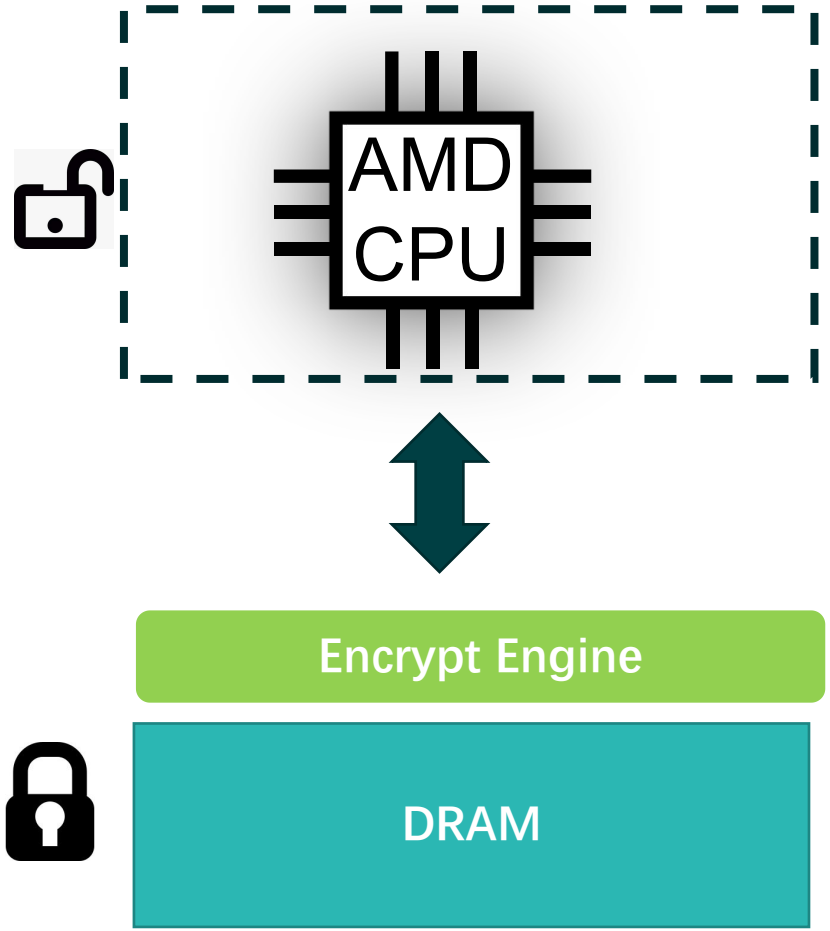


Chaining Modes: can not support random memory access in an efficient manner.

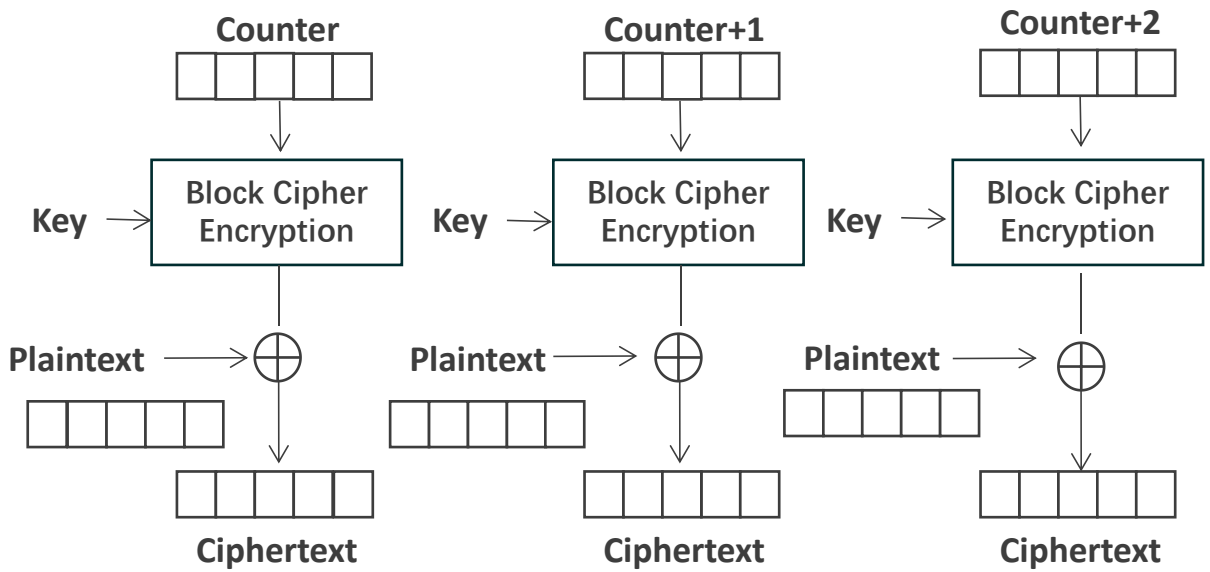


Cipher Block Chaining (CBC) mode encryption

Infeasible Encryption Modes



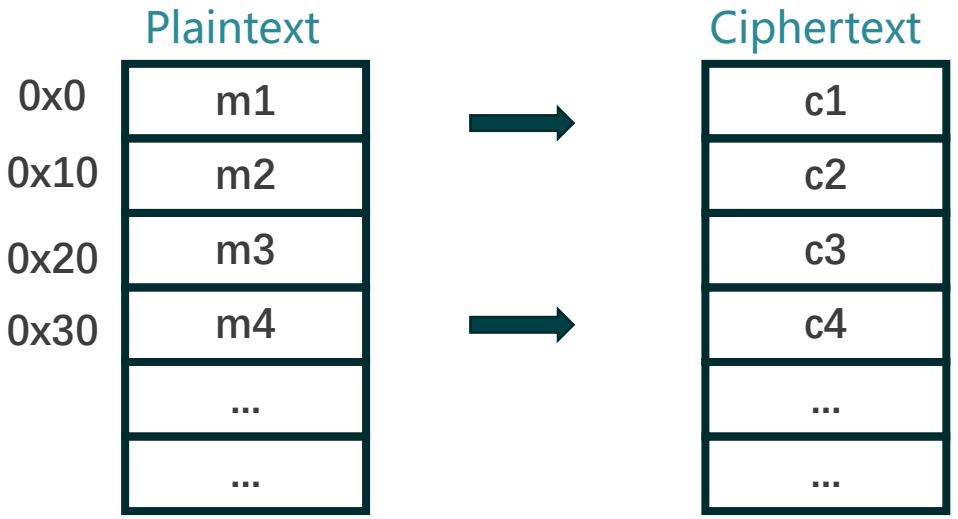
Freshness Modes: for large encrypted memory, additional space and lantency are need to maintain the counters.



Counter (CTR) mode encryption

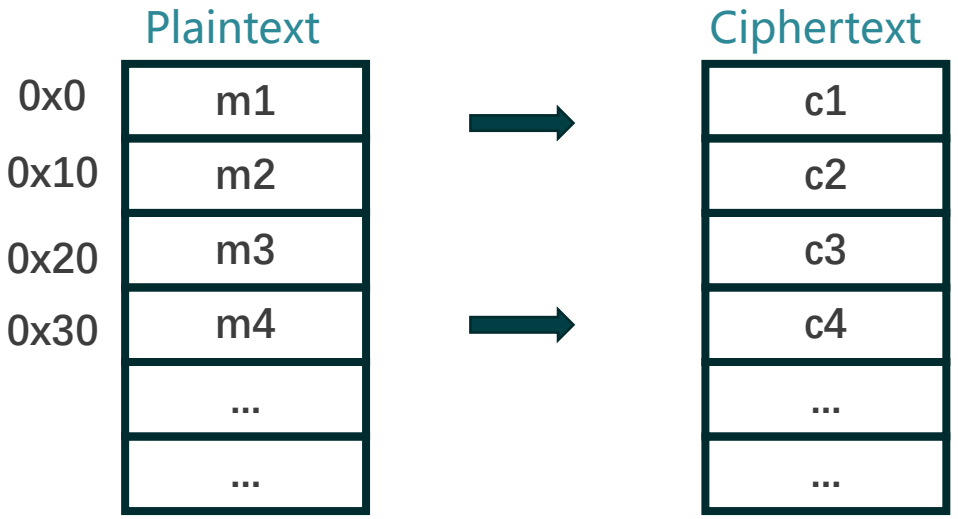
128-Bit AES Encryption with XEX Mode

Memory is independently encrypted per 128-bit block.



128-Bit AES Encryption with XEX Mode

Memory is independently encrypted per 128-bit block.



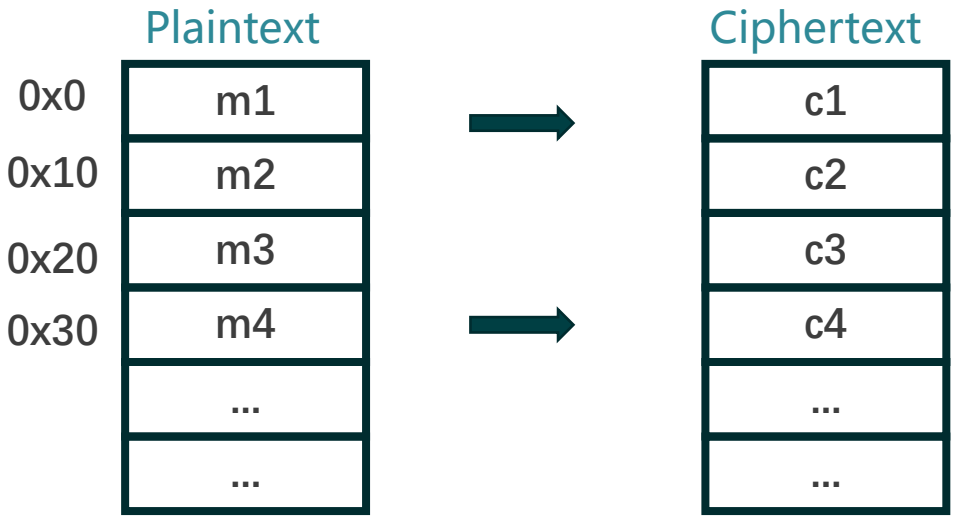
To avoid inferring plaintext (m) via the same ciphertext (C).

XEX mode with a tweak function T(x).

$$C = T(sPA_m) \oplus Enc(m \oplus T(sPA_m))$$

Ciphertext Side Channels

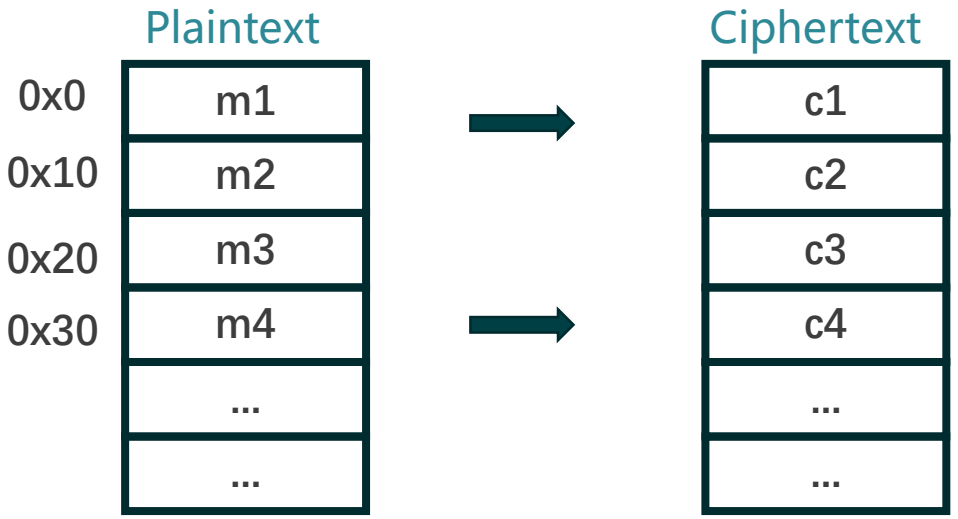
Memory is independently encrypted per 128-bit block.



The same plaintext at the same address is encrypted into identical ciphertext.

Ciphertext Side Channels

Memory is independently encrypted per 128-bit block.



The same plaintext at the same address is encrypted into identical ciphertext.

Deterministic Encryption:
AMD SEV
Intel TDX
Intel SGX on Ice Lake SP
ARM CCA

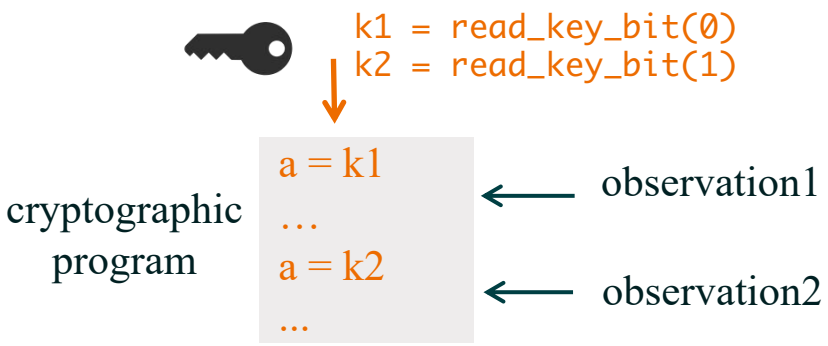
Inferring Relations of Key Bits

```
k1 = read_key_bit(0)
k2 = read_key_bit(1)
```

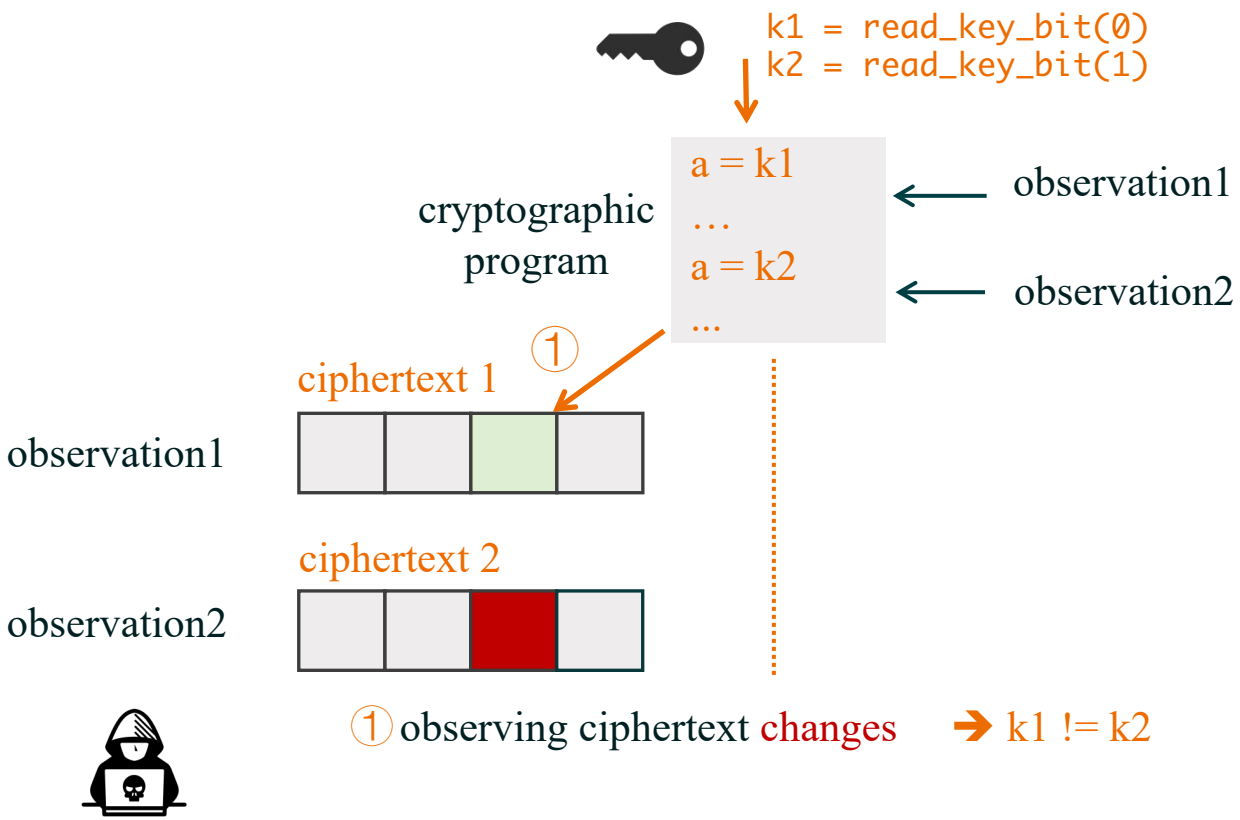
cryptographic program

```
a = k1
…
a = k2
...
```

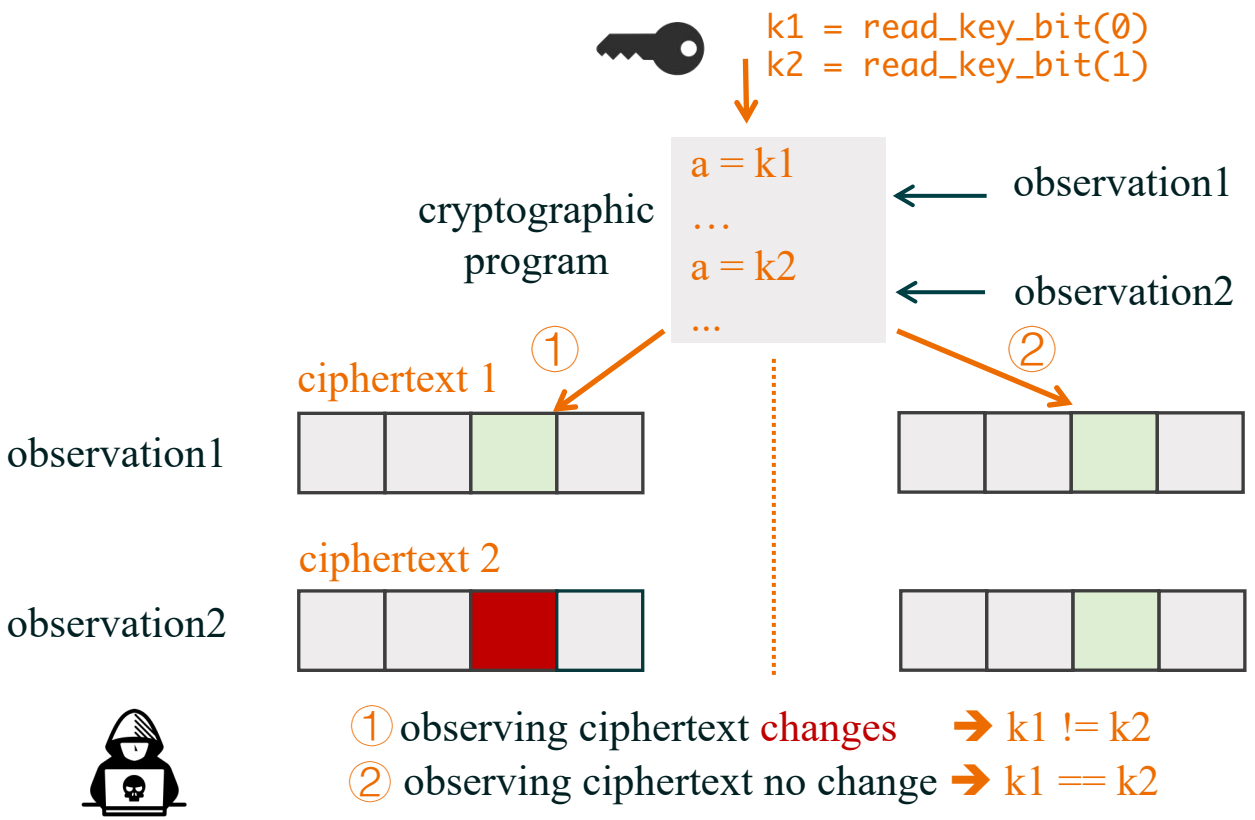
Inferring Relations of Key Bits



Inferring Relations of Key Bits



Inferring Relations of Key Bits



Hardware-level Mitigation

Change the memory encryption mode ...

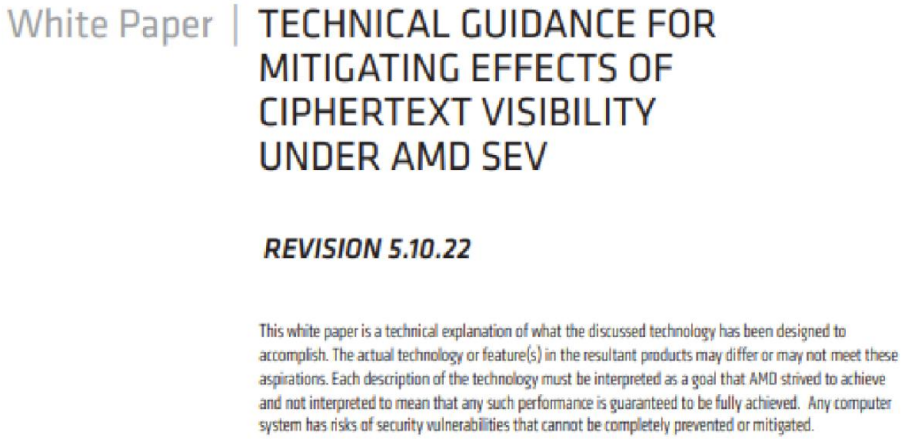
Hardware-level Mitigation

Change the memory encryption mode ?

too much performance overhead !

Software-level Mitigation

✓ **AMD has released a white paper to guide software developers in defending against ciphertext side channels.**


White Paper | TECHNICAL GUIDANCE FOR MITIGATING EFFECTS OF CIPHERTEXT VISIBILITY UNDER AMD SEV

REVISION 5.10.22

This white paper is a technical explanation of what the discussed technology has been designed to accomplish. The actual technology or feature(s) in the resultant products may differ or may not meet these aspirations. Each description of the technology must be interpreted as a goal that AMD strived to achieve and not interpreted to mean that any such performance is guaranteed to be fully achieved. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated.

- **1 . Data in Register**
- **2 . Data Padding**
- **3 . Data Masking**
- **4 . Data Moving**

Motivation

CipherH servers as a "vulnerability detector" to assist developers in assessing potential attack vectors of their software under ciphertext side channels.

Model Ciphertext Side Channels

- **Two sequential memory write operations.**
- **The two operations are secret-dependent.**

Model Ciphertext Side Channels

- **Two sequential memory write operations.**
- **The two operations are secret-dependent.**

- **Two writes:** $W_1()$ & $W_2()$
- **Two secrets:** k_1 & k_2
- **Written values:** $W_1(k_1)$ & $W_2(k_2)$

Model Ciphertext Side Channels

- **Two sequential memory write operations.**
- **The two operations are secret-dependent.**

- **Two writes:** $W_1()$ & $W_2()$
- **Two secrets:** k_1 & k_2
- **Written values:** $W_1(k_1)$ & $W_2(k_2)$

- **Safe Scenario 1:** $\forall k_1, k_2 \in K, W_1(k_1) = W_2(k_2)$
- **Safe Scenario 2:** $\forall k_1, k_2 \in K, W_1(k_1) \neq W_2(k_2)$

Model Ciphertext Side Channels

ciphertext keeps unchanged

$$\forall k_1, k_2 \in K, W_1(k_1) = W_2(k_2)$$

Safe

Model Ciphertext Side Channels

ciphertext keeps unchanged

$$\forall k_1, k_2 \in K, W_1(k_1) = W_2(k_2)$$

Safe

ciphertext changes

$$\forall k_1, k_2 \in K, W_1(k_1) \neq W_2(k_2)$$

Safe

Model Ciphertext Side Channels

ciphertext keeps unchanged

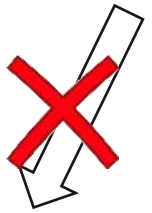
$$\forall k_1, k_2 \in K, W_1(k_1) = W_2(k_2)$$

Safe

ciphertext changes

$$\forall k_1, k_2 \in K, W_1(k_1) \neq W_2(k_2)$$

Safe



the change of ciphertext depends on k

$$\exists k_1, k_2, k_1', k_2' \in K, W_1(k_1) = W_2(k_2) \wedge W_1(k_1') \neq W_2(k_2')$$

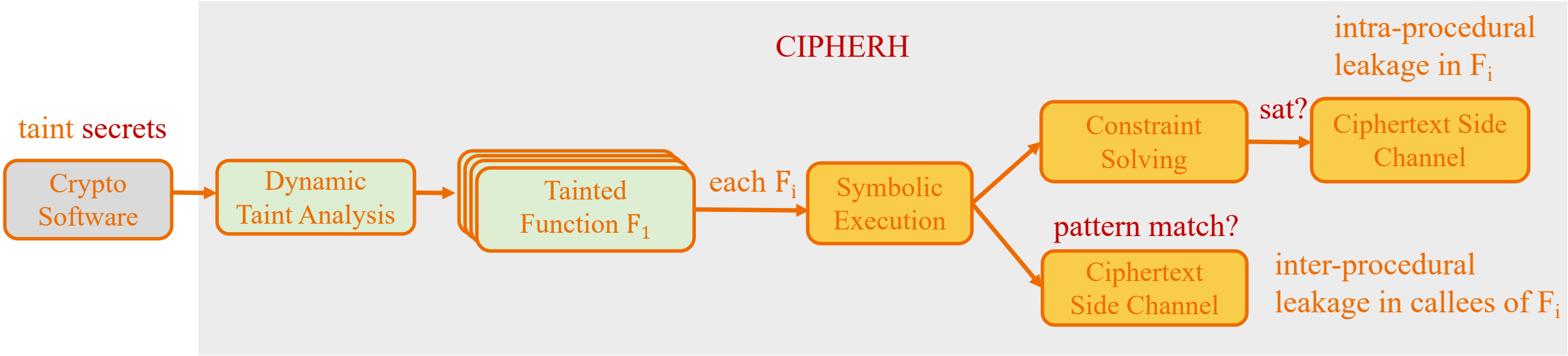
Model Ciphertext Side Channels

Information Leakage Scenario:

$$\exists k_1, k_2, k_1', k_2' \in K, W_1(k_1) = W_2(k_2) \wedge W_1(k_1') \neq W_2(k_2')$$

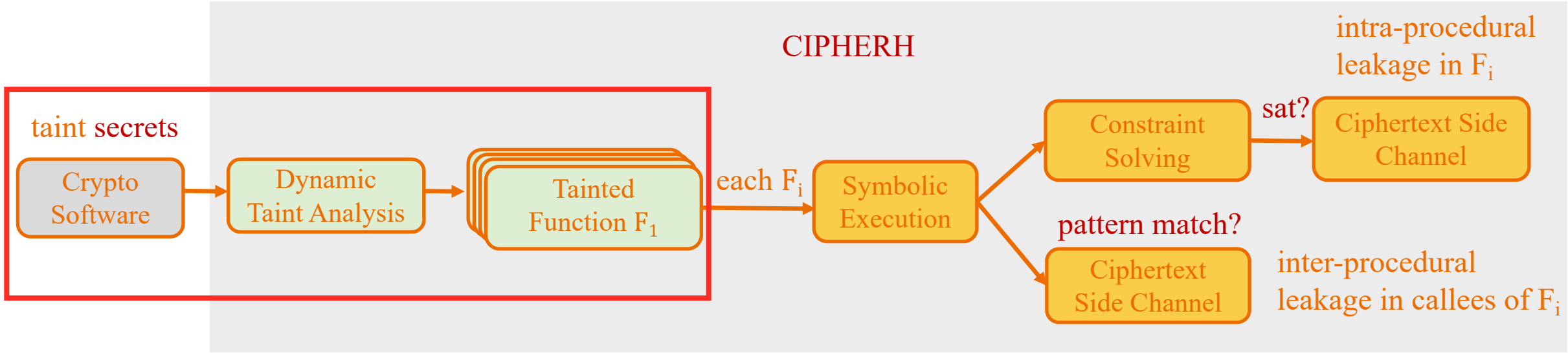
It models two different executions following the same path, such that during one execution, the second memory write operation changes ciphertext, whereas during the other execution, the second memory write operation retains the ciphertext.

Design



Dynamic Taint Analysis & Static Symbolic Execution

Dynamic Taint Analysis

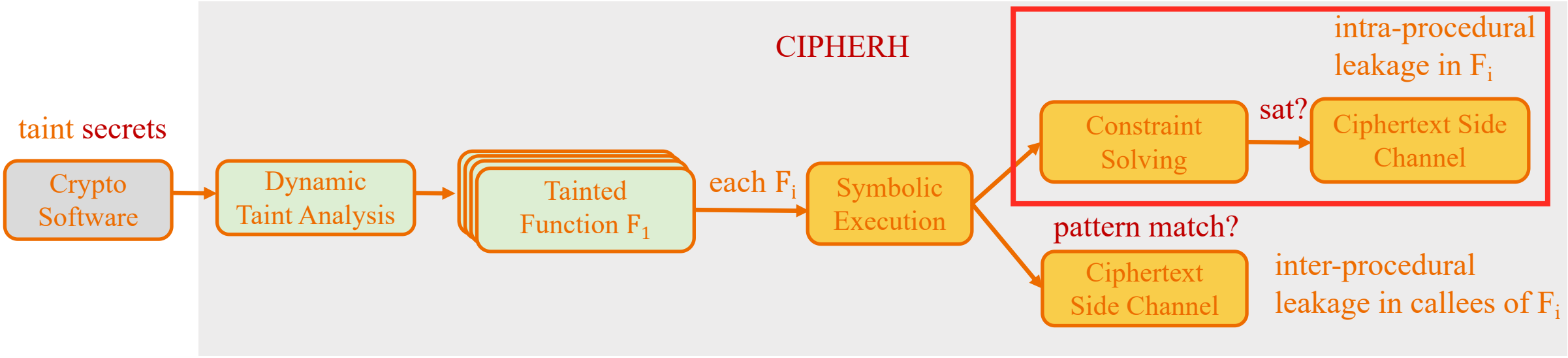


Taint source: secret

Taint propagation: based on DFSan

Taint sink: function parameter, return value, memory load

Intra-Procedural Symbolic Analysis

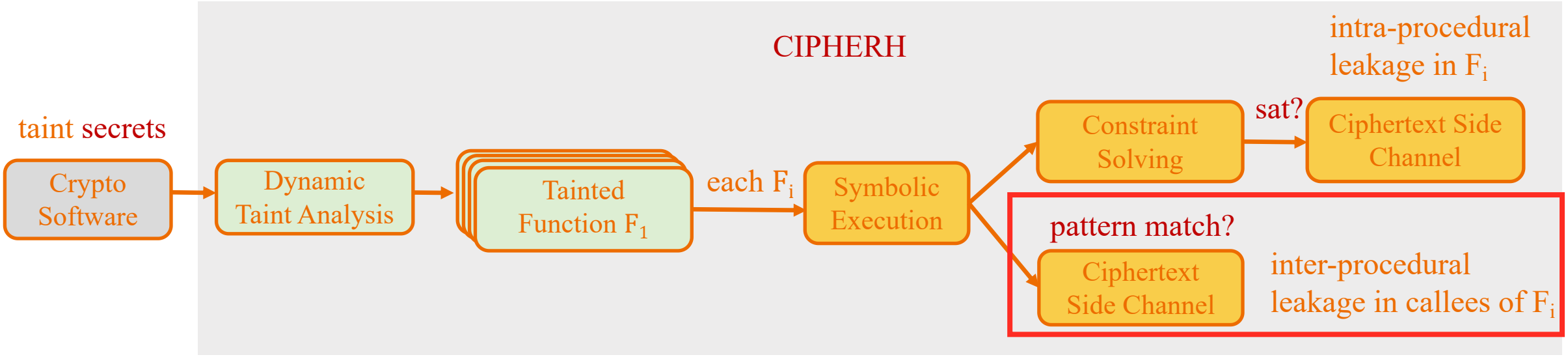


Memory lookup table: *M* and *W*

Constraint Solver: answer Yes or No

Outputs: two memory writes address and a pair of secrets

Inter-Procedural Symbolic Analysis



Pattern :

(1) *F* is repeatedly called by its caller function *Fc* at the same callsite.

(2) At least one input parameters of *F* is tainted.

Result

Implementation	Algorithm	Opt.	Intraprocedural Symbolic Execution		Interprocedural Symbolic Execution		Function
			(Vulnerable/Analyzed) Functions	Vulnerable Program Points	(Vulnerable/Analyzed) Functions	Vulnerable Program Points	(Tainted/Covered)
WolfSSL 5.3.0	ECDSA	-O2	3/53	6	1/2	12	53/92
WolfSSL 5.3.0	RSA	-O2	3/30	14	3/5	30	30/78
OpenSSL 3.0.2	ECDSA	-O3	4/68	6	4/11	29	68/1061
OpenSSL 3.0.2	RSA	-O3	9/142	53	11/38	55	142/1296
MbedTLS 3.1.0	ECDH	-O2	2/37	2	2/5	5	37/87
MbedTLS 3.1.0	RSA	-O2	2/39	2	4/7	22	39/83
Total			23/369	83	25/68	153	369/2697

Result

Implementation	Algorithm	Opt.	Intraprocedural Symbolic Execution		Interprocedural Symbolic Execution		Function
			(Vulnerable/Analyzed) Functions	Vulnerable Program Points	(Vulnerable/Analyzed) Functions	Vulnerable Program Points	(Tainted/Covered)
WolfSSL 5.3.0	ECDSA	-O2	3/53	6	1/2	12	53/92
WolfSSL 5.3.0	RSA	-O2	3/30	14	3/5	30	30/78
OpenSSL 3.0.2	ECDSA	-O3	4/68	6	4/11	29	68/1061
OpenSSL 3.0.2	RSA	-O3	9/142	53	11/38	55	142/1296
MbedTLS 3.1.0	ECDH	-O2	2/37	2	2/5	5	37/87
MbedTLS 3.1.0	RSA	-O2	2/39	2	4/7	22	39/83
Total			23/369	83	25/68	153	369/2697

- Use solutions k1, k2, k1', k2' to validate all intraprocedural Vul. Program points.

Result

Implementation	Algorithm	Opt.	Intraprocedural Symbolic Execution		Interprocedural Symbolic Execution		Function
			(Vulnerable/Analyzed) Functions	Vulnerable Program Points	(Vulnerable/Analyzed) Functions	Vulnerable Program Points	(Tainted/Covered)
WolfSSL 5.3.0	ECDSA	-O2	3/53	6	1/2	12	53/92
WolfSSL 5.3.0	RSA	-O2	3/30	14	3/5	30	30/78
OpenSSL 3.0.2	ECDSA	-O3	4/68	6	4/11	29	68/1061
OpenSSL 3.0.2	RSA	-O3	9/142	53	11/38	55	142/1296
MbedTLS 3.1.0	ECDH	-O2	2/37	2	2/5	5	37/87
MbedTLS 3.1.0	RSA	-O2	2/39	2	4/7	22	39/83
Total			23/369	83	25/68	153	369/2697

- Validate interprocedural results manully and confirm 144 out of 153 findings are ture positives.
- CipherH reports a vulnerable patch in function *mp_cond_swap_ct* in WolfSSL [1].

[1] The vulnerability has been fixed in WolfSSL version 5.4.0 onwards.

Efficiency Comparison

	Abacus [1]	CacheS [2]	CacheAudit [3]
RSA/OpenSSL	failed (in a few seconds)	failed	failed
RSA/MbedTLS	failed (in 7.3h)	failed	failed
ECDH/MbedTLS	timeout (> 18h)	failed	failed

None of them are scalable to analyze our test cases!

[1] Qinkun Bao, Zihao Wang, Xiaoting Li, James R Larus, and Dinghao Wu. Abacus: Precise side-channel analysis. ICSE, 2021.
[2] Shuai Wang, Yuyan Bao, Xiao Liu, Pei Wang, Danfeng Zhang, and Dinghao Wu. Identifying cache-based side channels through secret-augmented abstract interpretation. USENIX Security, 2019.
[3] Goran Doychev, Dominik Feld, Boris Kopf, Laurent Mauborgne, and Jan Reineke. CacheAudit: A tool for the static analysis of cache side channels. USENIX Security, 2013.

Pattern Influence

Inter-procedural findings for the ECDH/MbedTLS case by different patterns.

Pattern	Function (Vulnerable/Analyzed)	Inter-Procedural Vul. Program Points	False Positives
① & ②	2/5	5	1
①	8/19	17	10
Nil	11/35	21	14

The two patterns ① and ② are adequate for delivering a scalable inter-procedural analysis with convincing accuracy and low false positive rates.

Compiler Optimization

Inter-procedural findings for the ECDH/MbedTLS case by different patterns.

Optimization Options	Function Number (Vulnerable/Analyzed)	Intra-Procedural Vul. Program Points	Function Number (Tainted/Covered)
-O2	3/30	14	30/78
-O0	12/69	33	69/153

Aggressive optimization tends to place variables into registers, resulting in less memory writes and thus less vulnerabilities.

Conclusion

- Summary
  - CIPHERH formulates for the first time ciphertext side channels.
  - CIPHERH can identify ciphertext side channels in production software.
- Discussion
  - CIPHERH may produce false positives and false negatives.
  - Some tools may be developed to automate the elimination of vulnerable program points.

Thanks for Listening!

Sen Deng: 12032873@mail.sustech.edu.cn

Code: https://github.com/Sen-Deng/CipherH

Paper