Analysis of Software Countermeasures for Whitebox Cryptography

Subhadeep Banik[1], Andrey Bogdanov[2], Takanori Isobe[3], Martin B. Jepsen[2]

[1] Temasek Labs, Nanyang Technological University, Singapore
[2] DTU Compute, Technical University of Denmark, Lyngby
[3] Sony Corporation, Tokyo, Japan

bsubhadeep@ntu.edu.sg

FSE 2017, Tokyo, Japan

June 13, 2018

Overview

Whitebox Cryptography

- What is Whitebox cryptography ?
  - Informally, a software implementation of a cryptosystem
  - Secret Key is usually embedded in the implementation
  - User can use it to encrypt/decrypt data
  - But he must not be able to deduce the secret key.
- Original use case: DRM for example in streaming content
  - Legitimate user should be able to decrypt and view content
  - But not deduce secret key to distribute

Whitebox Encryption

- Cryptography designed to be secure in a hostile environment.
- Whitebox model: Adversary has
  - Full control over execution environment.
  - Can perform analysis over executable/binary
  - Can alter/inspect memory
  - Can change intermediate results
- Goal is to guarantee the security of the secret key in such a setting.

Whitebox Schemes

- Seminal whitebox scheme for AES proposed by Chow et al. [SAC02]
- AES operations represented by a series of Lookup Tables
- However the scheme has already been broken.
  - Billet et al [SAC 04], Lepoint et al [SAC13]
  - Saneflix et al [Blackhat 15] (DFA)
  - Needs Read/Write access exact locations of tables
  - Reverse engineer the system by algebraic attack.
- However a simple control flow randomization prevents the attacks.
- Randomize table accesses in each round.

Chow's Whitebox Scheme for AES

Chow et al's Whitebox AES implementation: Top Level View

Algorithm 1

$S^{(0)} \leftarrow \texttt{Plaintext}$
for $r = 1$ to 10 **do**
  $\hat{S}^{(r-1)} \leftarrow \texttt{ShiftRow}(S^{(r-1)})$
  for $j = 0$ to 3 **do**
    $(s_{0,j}^{(r)}, s_{1,j}^{(r)}, s_{2,j}^{(r)}, s_{3,j}^{(r)}) \leftarrow \text{EnSubround}_j^{(r)}(\hat{s}_{0,j}^{(r-1)}, \hat{s}_{1,j}^{(r-1)}, \hat{s}_{2,j}^{(r-1)}, \hat{s}_{3,j}^{(r-1)})$
  end for
end for
$\texttt{Ciphertext} \leftarrow S^{(10)}$

Top view of EnSubround

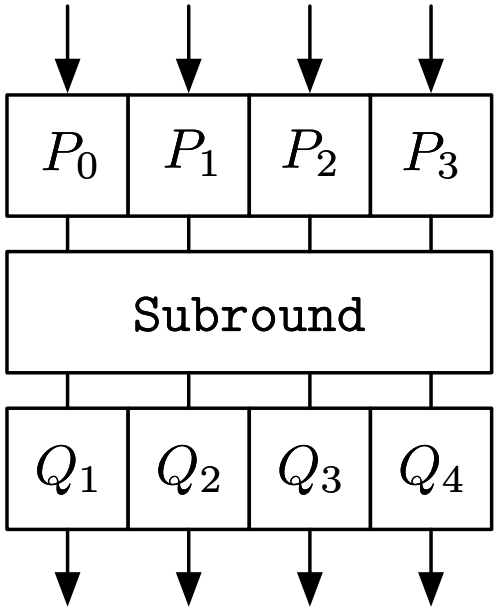


Figure: High-level view of an encoded subround of the Chow et. al. AES.

- P_i, Q_i arbitrary 8 bit bijections
- Pairwise cancelling i.e. $P_i^{-1}[r+1] = Q_i[r]$
- Loosely speaking Subround performs the functionality of 1 AES round.

Composition of Encoded Subround

- Level 1 T-boxes (Table A):
  - Combine the effect of ARK, SB, SR and a part of Mixcolumns.
  - Each round has 16 T-boxes that maps 4 bits $\rightarrow$ 32 bits.
  - Additional Mixing bijection to improve diffusion
- Level 2 Xor-tables (Table C1):
  - Does Xor operations in Mixcol not done in level 1.
- Level 3 T-boxes (Table B):
  - Partially cancels the Mixing Bijection in Level 1.
- Level 4 Xor-tables (Table C2):
  - Does Xor operations in Mixcol not done in level 3.

Figure

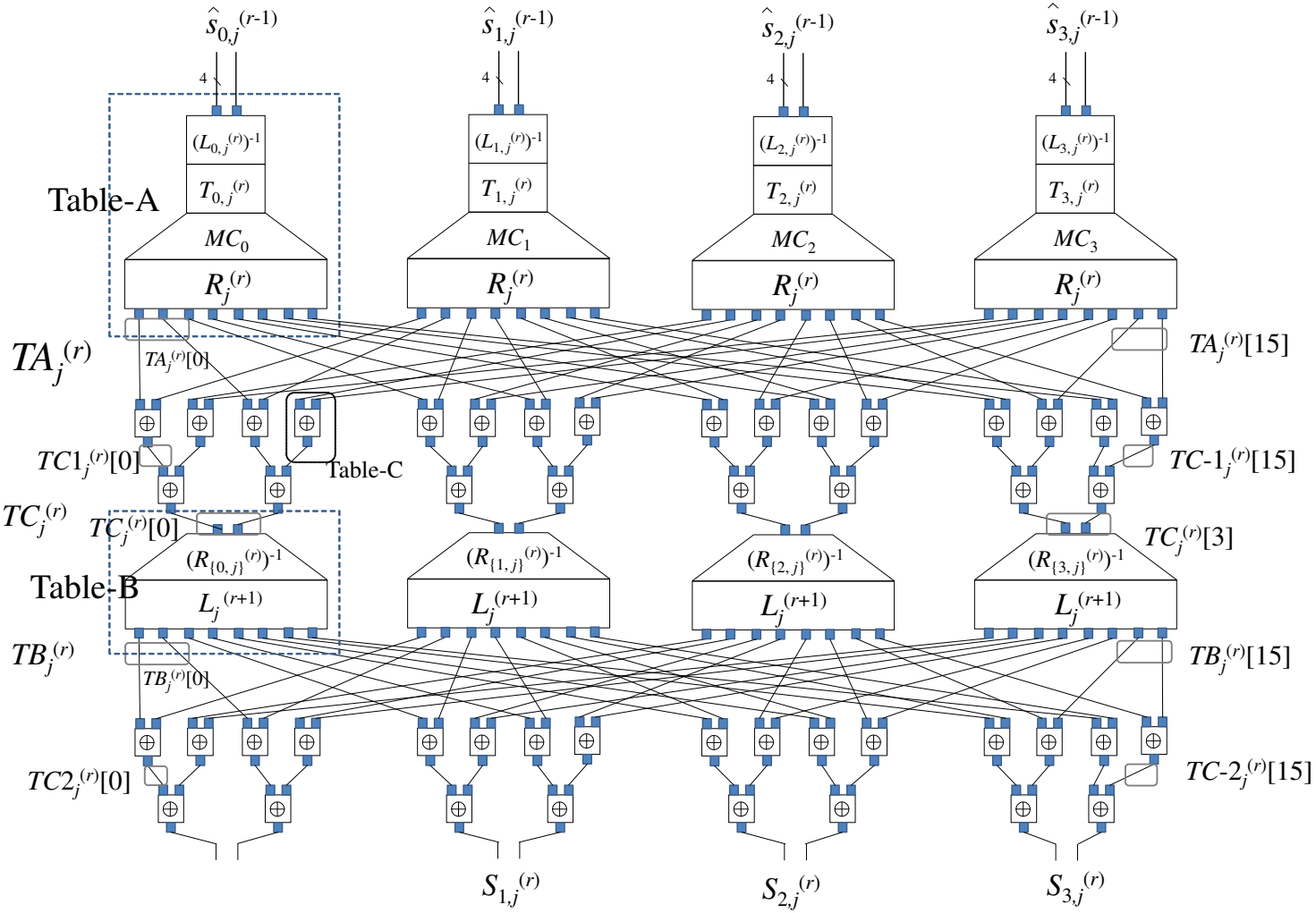
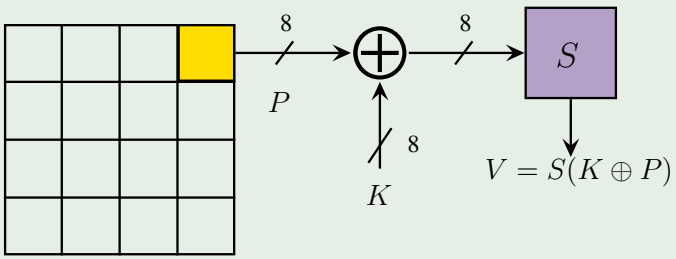


Figure: Table-based composition of the encoded subround of the Chow et. al. AES.

DCA Attacks

- Proposed by Bos et al [CHES 16] and Saneflix et al [Blackhat 15]
- Software equivalent of DPA attacks in Hardware.

Example

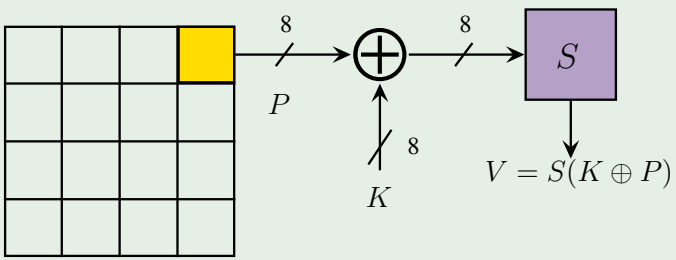


- Power consumption follows hamming weight model
- Take Power traces $\mathcal{P}_i(t)$ for numerous plaintexts PT_i
- Eavesdrop on V: Power likely to be $\propto HW(Sb(PT \oplus K))$

DCA Attacks

- Proposed by Bos et al [CHES 16] and Saneflix et al [Blackhat 15]
- Software equivalent of DPA attacks in Hardware.

Example



- For each guess K_j, plaintext PT_i , compute $H_{ij} = HW(Sb(PT_i \oplus K_j))$
- If $K_j = K \Rightarrow$ for some t_0, $\mathcal{P}_i(t)$ would be perfectly correlated with H_{ij}.
- t_0 usually denotes the time instant when the S-box value is computed.

DCA Attacks in Whitebox Scheme

- The whitebox binary is instrumented using PIN/Valgrind.
  - This enables inserting code in runtime
- Every memory operation is instrumented so that it records the values and addresses of the operation reads or writes.
- Memory Address Space $\rightarrow$ Used to store LUTs that make up the whitebox scheme.
- Each memory address leaks the values of inputs to the tables.
  - This signal can be used as a Power trace to mount DPA.

DCA Attacks in Whitebox Scheme

- Two types of DCA possible
  - Address based DCA (ADCA): Uses memory addresses from software traces.
  - Value based DCA (VDCA): Uses values read from memory addresses.
- The original paper [Bos et al CHES 16] only talks about ADCA
- One can just as easily use value traces to do the attack.

Countermeasures

- We will discuss three countermeasures that can be applied to the binary
  - Control Flow Obfuscation
  - Table randomization
  - Adding Dummy operations
- We will see how each countermeasure is designed to thwart a particular class of attack.

Control Flow Obfuscation

- Attacks in Billet et al [SAC 04], Lepoint et al [SAC13], Saneflix et al [Blackhat 15] (DFA)
- Needs Read/Write access to exact locations of tables
  - Can be prevented if we randomize the order of table look ups for each round/encryption
  - Attacker can no longer read exact internal bytes for different encryptions.
- Make a dependency graph first to determine order of accesses.

Control Flow Obfuscation

Example of Dependency graphs

```
1 v0 = table_0[v1];
2 v1 = table_1[v0];
3 v2 = table_2[v0];
4 v0 = table_3[v3];
```

table_1 and table_2 can be accessed in any order

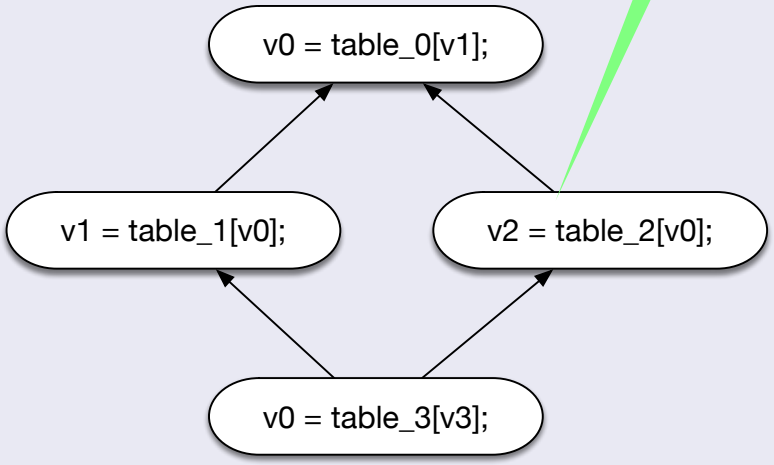


Figure: Dependency graph of example program

Table Randomization

Countering ADCA

- LUTs are linearly arranged in memory space
- Hence address reads reveal table inputs
- What if we randomize??

Won't thwart VDCA:(

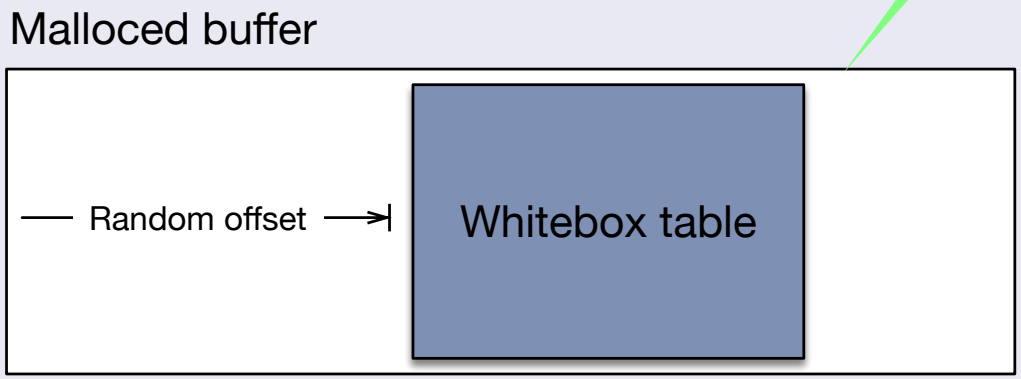


Figure: Random offset of table in memory.

Dummy Operations

Random disarrangement - Mangard [CTRSA 04]

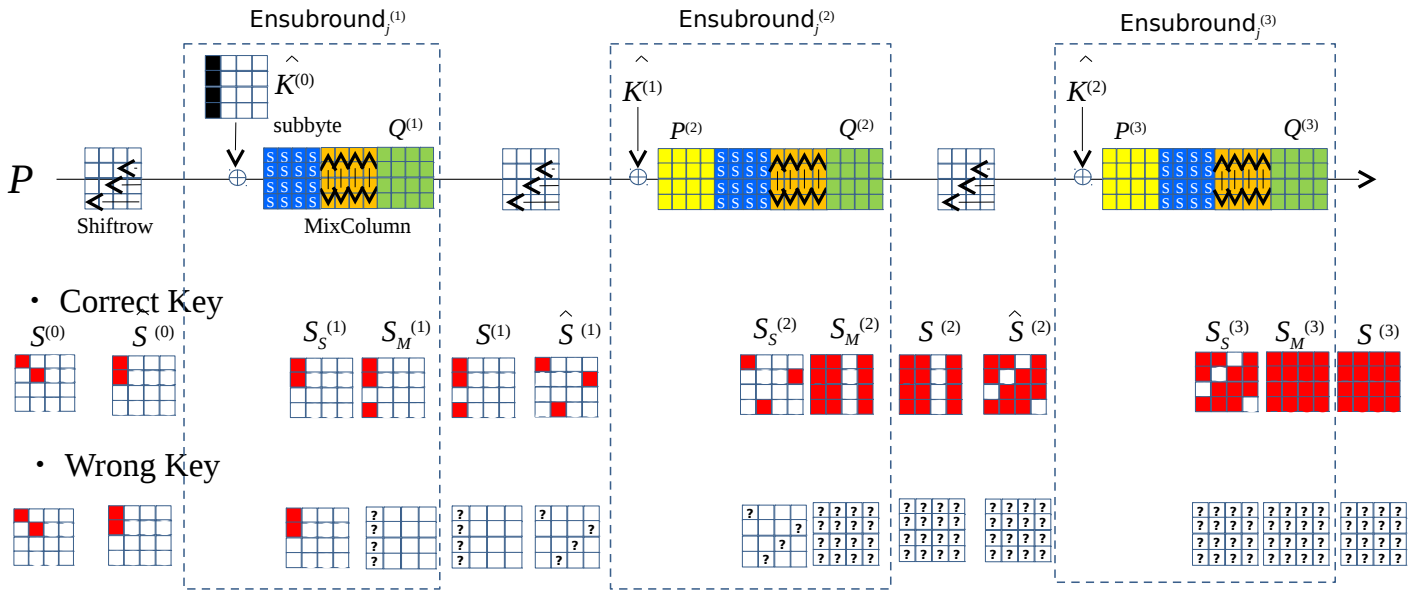
- For correct key guess K_j for some t_0, $\mathcal{P}_i(t)$ would be correlated with H_{ij}
- Randomize time at which $V = S(PT \oplus K)$ is computed for each encryption
- As a result, the time instances at which V is computed no longer align with each other over multiple traces.
- And so it becomes more difficult to mount a power attack.
- In software this amounts to dummy table lookups to add noise.

Listing 1: Example of dummy lookups

```
1 for (int i = 0; i < rand() % 16; i++)
2       dummy = table_0[rand() % 128];
3 v0 = table_0[v1];
```

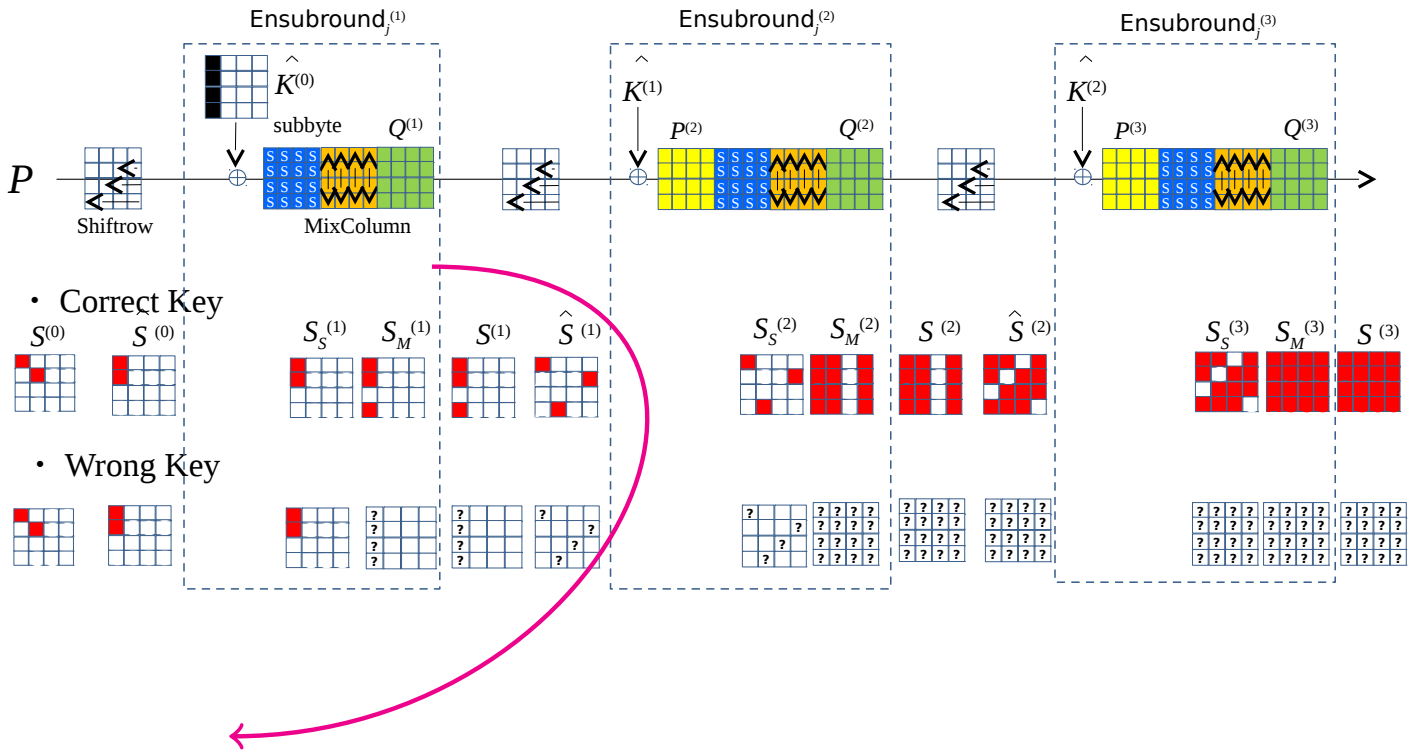
Zero Difference Enumeration Attacks

- An attack for whitebox binaries with control flow obfuscation.
- Standard differential attack, with key guessing.
- Query with key dependent β-plaintext pairs.



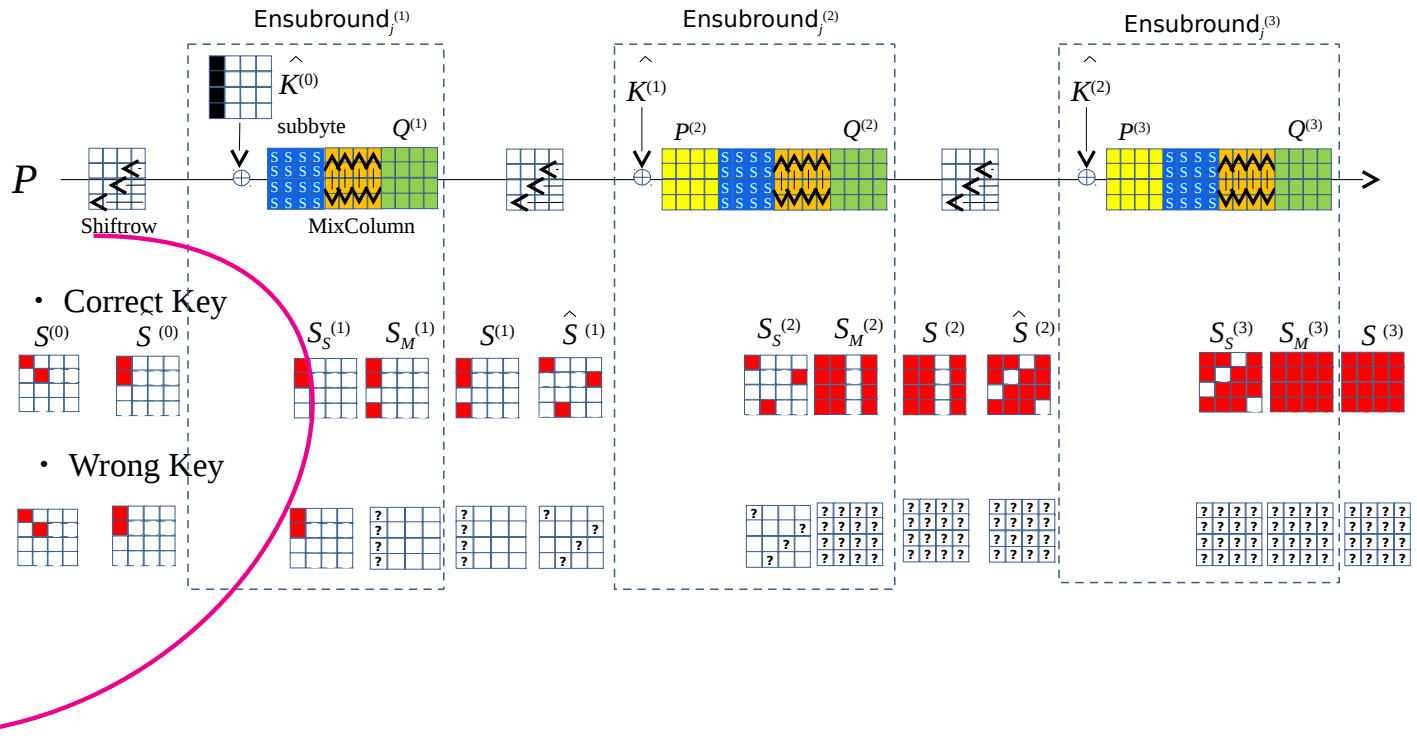
Zero Difference Enumeration Attacks

- Let the state after 1st Mixcol be S_M
- Choose a byte Δ, and a column $S_M[1]$ randomly.
- So that the column $S_M[2] = S_M[1] \oplus [\Delta, 3\Delta, 0, 2\Delta]$



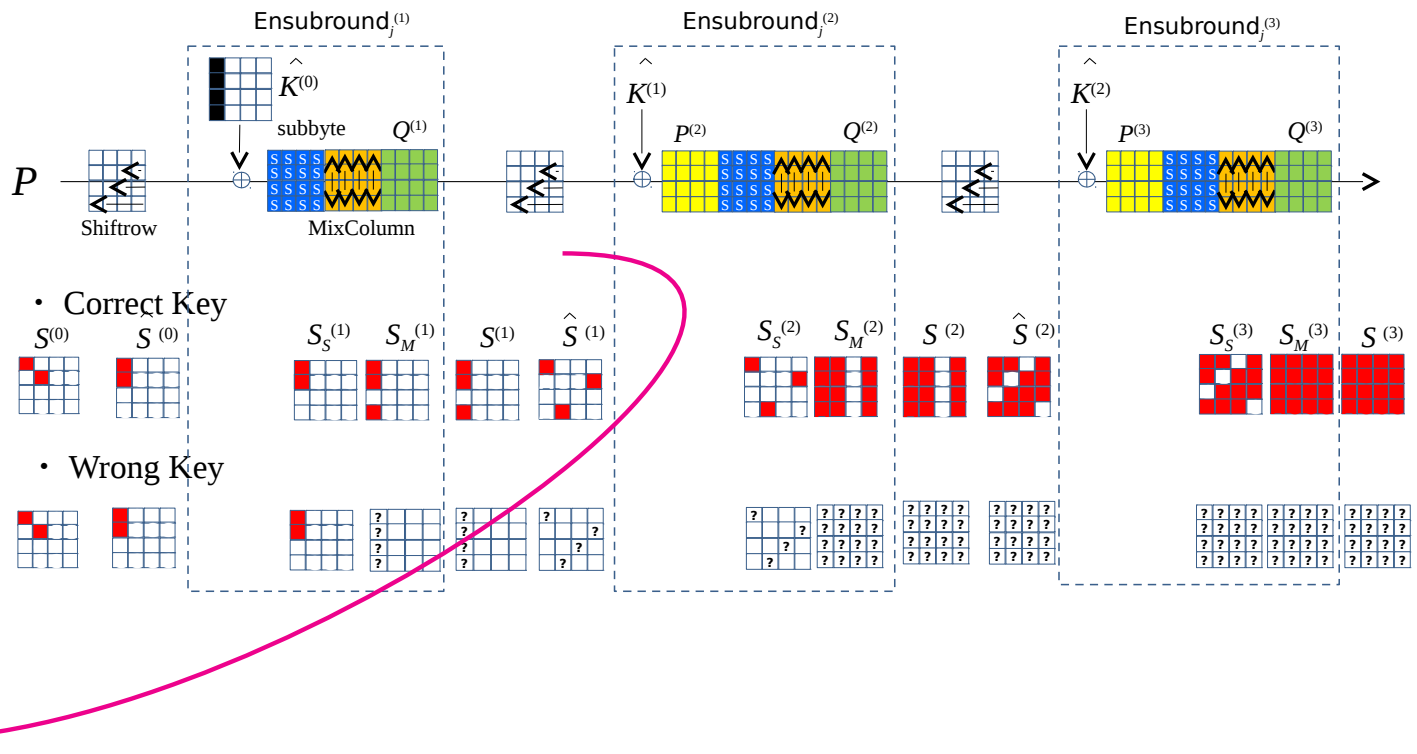
Zero Difference Enumeration Attacks

- Invert Mixcol, Shiftrow and Subbyte layers and guess keys k_0, k_5
- Thus we get values of indices 0, 5 of the plaintext pairs.
- Remaining 14 indices are filled with any constants.



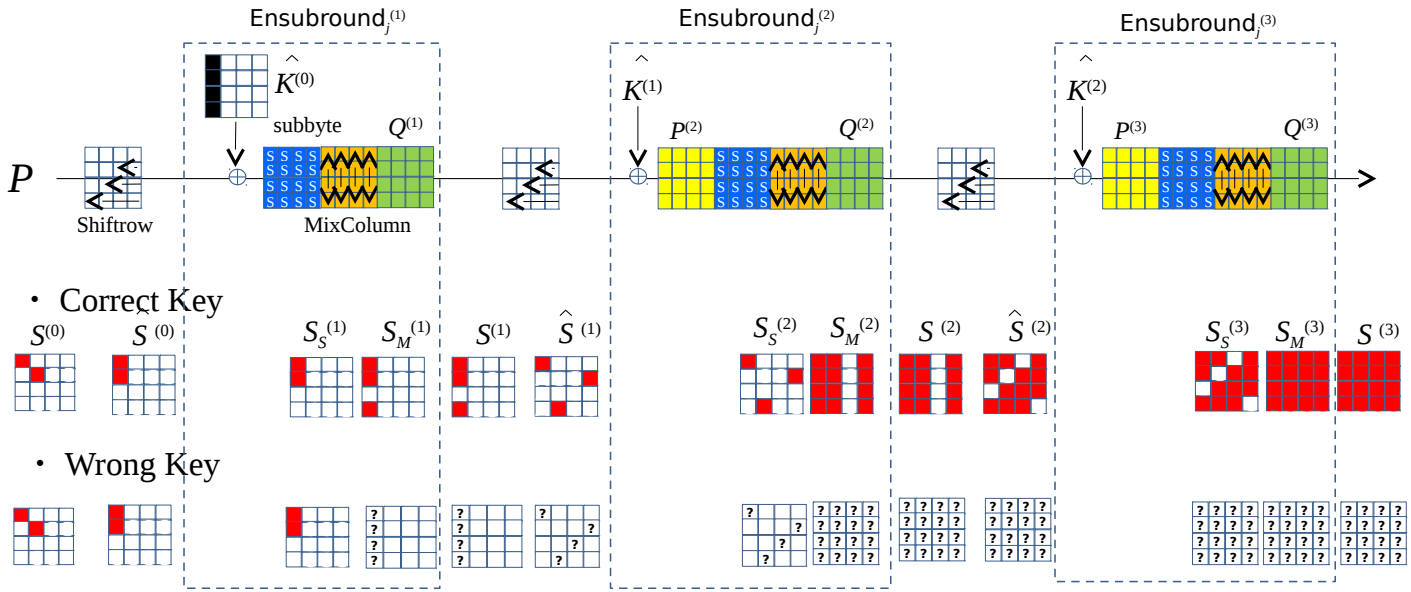
Zero Difference Enumeration Attacks

- Query for encryption of plaintext pairs.
- If the guessed key is equal to the key in whitebox:
- A lot of internal bytes in both plaintext pairs are same



Zero Difference Enumeration Attacks

- For correct guess 307 internal bytes are guaranteed to be equal in first 3 rounds. (Incorrect guess: only 244 bytes)
- Thwarts flow obfuscation+table randomization



CHES 2016 Challenge

- As a part of CHES 2016 challenge, a C implementation and compiled binary of the Chow whitebox was provided.
- Goal was to find the key.
- We applied the attacks we have developed both before and after applying the countermeasures.
    - The C code has 4048 static tables
    - We parsed the code and made a data dependency graph.
    - Keeping dependencies in mind, the flow was randomized.
    - Other countermeasures also incorporated.

Dependency Graph

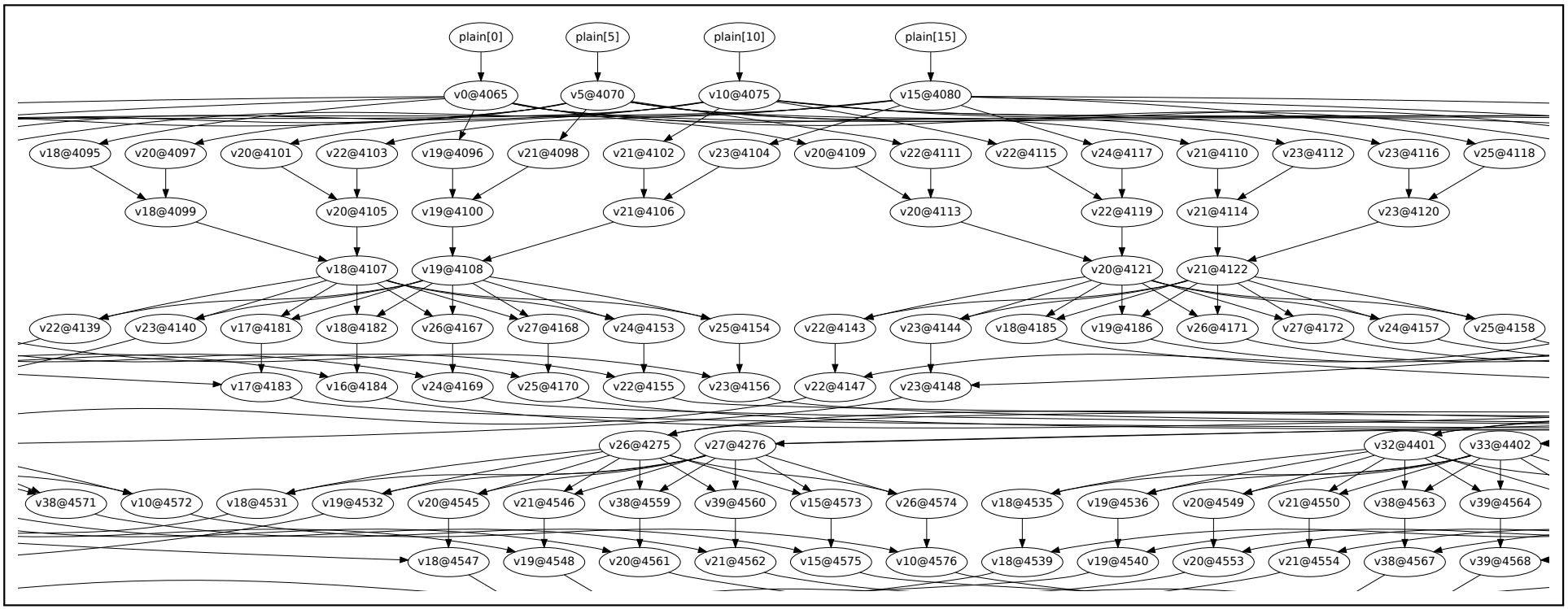


Figure: Data flow in the CHES 2016 challenge. The graph shows how the data flows from 4 bytes of the input plaintext through the tables of the CHES 2016 challenge whitebox. Each node represents a line in the source code, and is annotated with the variable that is modified followed by the line number. What is seen is part of a masked MixColumns step on nibbles, followed by masked xor tables to combine the results.

DCA results

Table: DCA on the CHES 2016 challenge.

Attack type	Countermeasures	Time per trace	Size per trace	# Traces	# Key bytes (found in top 10)	# Key bytes (found as best)	Correlation time (h:m:s)
ADCA	None	0.65 seconds	19,248 bytes	4,000	16	15	48:36
	Shuffling	3.14 seconds	6,224 bytes	4,000	13	12	16:22
				10,000	13	12	40:06
	Shuffling and random offsets	3.35 seconds	51,680 bytes	10,000	0	0	6:41:59
	Dummy operations	4.61 seconds	38,768 bytes	4,000	1	0	1:41:09
VDCA	None	0.65 seconds	19,248 bytes	4,000	11	4	49:15
	Shuffling	3.14 seconds	6,224 bytes	4,000	5	0	16:34
				10,000	5	1	41:08
	Shuffling and random offsets	3.35 seconds	40,448 bytes	10,000	13	12	5:15:57
	Dummy operations	4.61 seconds	38,768 bytes	4,000	0	0	1:49:29

Metrics for DCA on the CHES 2016 whitebox challenge with countermeasures. The time per trace is the total time to record and store one execution trace. Address and value traces are recorded at the same time, which is why the time per trace is equal in ADCA and VDCA. The size per trace is the bytes of storage used per execution trace, when recording 1/3 of the encryption function with appropriate filtering (see subsections). We record the number of correct key bytes that are ranked in the top 10 and best position according to the correlation value. The time is the total time to run the DPA correlation tool on the traces, excluding tracing.

ZDE results

Table: ZDE on the CHES 2016 challenge.

Attack type	Countermeasures	Time per trace	Size per trace	# Traces	# Key bytes found	Total time (h:m:s)
ZDE	None	0.000012 seconds	2,048 bytes	$500 \cdot 2^{17}$	2	0:3
	Shuffling	0.001641 seconds	2,048 bytes	$500 \cdot 2^{17}$	2	7:0
	Shuffling and random offsets	0.003594 seconds	2,048 bytes	$500 \cdot 2^{17}$	2	15:20
	Dummy operations	0.071543 seconds	4,096 bytes	$5000 \cdot 2^{17}$	$0^{\star}$	5:5:15

Metrics for ZDE on the CHES 2016 whitebox challenge with countermeasures. The attack was run as 256 parallel instances on a cluster, and the final scores of candidate keys was filtered to identify the top choice. The time is the total time to run the attack, including tracing. Only 2 key bytes were attacked. $\star$: The correct key bytes were ranked as number 2.

Conclusion

- We study attacks and countermeasures for Chow's whitebox scheme
- Dummy operations seem to be the most effective countermeasure- but only just.
- Further research direction: Differential Fault attacks.

Thank You