Towards Easy Key Enumeration

Changhai Ou, Degang Sun, Zhu Wang, Xinping Zhou, and Juan Ai

[1] Institute of Information Engineering, Chinese Academy of Sciences
[2] School of Cyber Security, University of Chinese Academy of Sciences
Email:ouchanghai@iie.ac.cn

Abstract. Key enumeration solutions are post-processing schemes for the output sequences of side channel distinguishers, the application of which are prevented by very large key candidate space and computation power requirements. The attacker may spend several days or months to enumerate a huge key candidate space (i.e. 2^{40}). In this paper, we aim at pre-processing and reducing the key candidate space by deleting impossible key candidates before enumeration. A new distinguisher named Group Collision Attack (GCA) is given. Moreover, we introduce key verification into key recovery and a new divide and conquer strategy named Key Grouping Enumeration (KGE) is proposed. KGE divides the huge key space into several groups and uses GCA to delete impossible key combinations and output possible ones in each group. KGE then recombines the remaining key candidates in each group using verification. The number of remaining key candidates becomes much more smaller through these two impossible key candidate deletion steps with a small amount of computation. Thus, the attacker can use KGE as a pre-processing tool of key enumeration and enumerate the key more easily and fast in a much smaller candidate space.

Keywords: key enumeration, KGE, Group Collision Attack, DPA *contest v4*, divide and conquer, side channel attack

1 Introduction

Side channel attacks make complex key recovery simple by using divide and conquer. Divide and conquer attacks, such as correlation power analysis (CPA) [6], differential power analysis (DPA) [11], template attack (TA) [7], mutual information analysis (MIA) [9], etc., divide the full key into several pieces and conquers each of them. If enough power traces are used, the correct sub-key bytes are on the top of sequences output by distinguishers. By using divide and conquer, the complex key recovery becomes simple. For example, if the attacker divides the 16 bytes sub-key of AES into 16 chunks and conquers them one by one, the amount of computation is reduced from 2^{128} to $2^8 \cdot 16$. However, if he doesn't have enough power traces, it's possible that one or several sub-key bytes are not ranked on the top of their corresponding sequences, but somewhere close to the top. The attacker has to use key enumeration solutions to enumerate the key candidates or use key rank estimation solutions to evaluate the security level.

Recently, several key rank estimation solutions are proposed to gauge the security level of implementations for which enumeration is beyond reach [28, 12, 26, 21]. Solutions such as [2, 10, 13] typically allow estimating the rank of a 128- or 256-bit key with an accuracy of less than one bit, within second of computation. Several key rank estimation solutions are even compared in [20]. However, unlike key enumeration, key rank estimation is considered as an evaluation tool, since it requires knowledge of the master key, which enables the evaluators to approximate the security level of the cryptographic implementation, specifically, by approximating the position of the master key.

Both key enumeration and key rank estimation are post-processing tools of side channel attack outcomes. Compared to key rank estimation, key enumeration in [21] is defined as an adversarial tool, since it allows to test key candidates without knowledge of the master key [19, 25, 4]. However, key enumeration is limited to the computational power of the evaluator [26]. That is, the only leaking devices for which we can evaluate the security are the ones that are practically insecure (i.e. for which the leakage allows key enumeration). To enumerate a key space 2^{40}, several days or months are needed. Moreover, large memory also prevents the application of these solutions.

In this paper, we aim at reducing the key candidate space. For example, from 2^{60} to 2^{20}. Since a wrong candidate ranking in the first several places of the outputs of a distinguisher may not rank in the first several places of the outputs of another distinguisher. Though combining different distinguishers, some of these candidates can be deleted. By doing this, the attacker can enumerate the key in a smaller candidate space. In order to achieve this goal, we need to pre-process the key candidate space before enumeration and delete a part of impossible key candidates. Here we use collision attack [23, 5, 14] to post-process the outputs of CPA, which attempts to establish the relationship between different key bytes by collisions, such as "**test of chain**" proposed by Bogdanov et al [3]. Each chain includes one or several pairs of collisions. They used two thresholds, one is for the key and another for $\Delta_{(k_a,k_b)} = k_a \oplus k_b$ between two key bytes k_a and k_b. Here, k_a and k_b denotes the a-th and b-th key chunks. They tried to find a long chain from k_1 to k_{16} including 16 steps.

Wang et al. proposed fault tolerant chain (FTC) in [27], which was another practical scheme of key recovery in a large candidate space (i.e. 2^{64}). In this paper, we only consider AES-256. So, the length of each key chunk is 8 bits. However, any chunk falling outside the threshold Thr_k will result in very complex or even fail key recovery in FTC. Changhai et al. proposed group verification chain in [18], which enhanced FTC significantly. They used several key bytes to verify one key bytes. The frequency or weight of the correct key byte values is higher than these of wrong ones. However, this scheme is somewhat a kind of key re-ordering. It is not a good choice to use it for key enumeration.

So, how to fast enumerate the correct key in a large space far beyond the computational power of evaluator if exhaust attacks being used is still worthy of further research. In order to better solve this problem, we combine the advantages of full key recovery and divide and conquer attacks, and propose a new

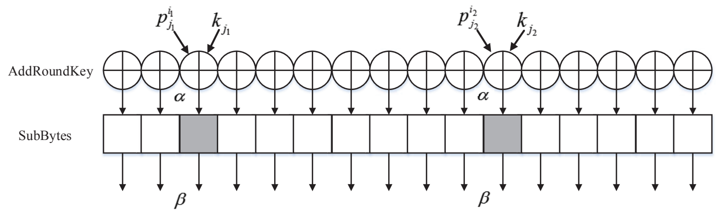

Fig. 1. A linear collision for two AES executions.

divide and conquer solution named Key Grouping Enumeration (KGE)in this paper. KGE divides the entire key of AES-256 into several big groups (pieces) and use a new distinguisher named Group Collision Attack (GCA) to post-process the outcomes of distinguishers (i.e. CPA) to delete the impossible key combinations in each group. KGE then uses verification chain to delete impossible key combinations among groups. The remaining key candidates are greatly reduced after these two impossible key combination deletion steps. The total amount of computation and memory requirements of KGE is very small. Our KGE can run on a common desktop computer and quickly delete impossible key candidates before enumeration. The attacker then can enumerate the key in a new candidate space further smaller than the original one.

2 Preliminaries

2.1 Collision Attack

Bogdanov et al. introduced linear collision attack in [3]. AES performs the SubBytes operation (16 parallel S-box applications) in the first round. A generalized internal AES linear collision occurs if there are two S-boxes in the same AES encryption or several AES encryptions accepting the same byte value as their input (as shown in **Fig. 1**). $K = \{k_j\}_{j=1}^{16}$, $k_j \in F_{2^8}$ is the 16-byte subkey in the first round of AES. AES plaintexts are denoted by $P^i = \left\{p_j^i\right\}_{j=1}^{16}$, $p_j^i \in F_{2^8}$, where i=1,2,... is the number of AES executions.

If a collision

$$S(p_{j_1}^{i_1} \oplus k_{j_1}) = S(p_{j_2}^{i_2} \oplus k_{j_2}) \tag{1}$$

happens within the first round of AES (as shown in **Fig. 1**), the attacker obtains a linear equation

$$p_{j_1}^{i_1} \oplus p_{j_2}^{i_2} = k_{j_1} \oplus k_{j_2} = \Delta_{(k_{j_1}, k_{j_2})}. \tag{2}$$

If m collisions are detected, then a system of m linear equations can be obtained:

$$\begin{cases} k_{j_1} \oplus k_{j_2} = \varDelta_{(k_{j_1}, k_{j_2})}, \\ k_{j_3} \oplus k_{j_4} = \varDelta_{(k_{j_3}, k_{j_4})}, \\ \qquad \vdots \\ k_{j_{2m-1}} \oplus k_{j_{2m}} = \varDelta_{(k_{j_{2m-1}}, k_{j_{2m}})}. \end{cases} \tag{3}$$

It is worth noting that some of these equations are independent. Thus they can be divided into h_0 independent subsystems with respect to the parts of key [3], of which each may have one free variable and one or more equations. Let h_1 denotes the number of all missing variables which are not in these subsystems. Each of the subsystems or missing variables is called a chain. Each equation is defined as a step of a chain. Hence the number of chains $h = h_0 + h_1$.

2.2 Test of Chain

Bogdanov et al. defined test of chain in [3]. Suppose that the attacker uses CPA to calculate the correlation coefficients for each key candidate. He sorts all 256 key byte candidates in descend order according to their corresponding correlation coefficients. He obtains the 16 guessing key byte sequences $\{\xi_i | i = 1, 2, \cdots, 16\}$ of AES algorithm. He also uses Correlation-enhanced Collision Attack (CCA) to calculate the correlation coefficients of $\varDelta_{(k_a, k_b)}$.

Chain ξ of length n consisting of key-byte indices $j_1, \cdots, j_n$. In each list ξ_i, they only consider values among the top m positions. They are the most possible candidates of the key byte k_i. The attacker tries to find a chain from ξ_1 to ξ_{16} including 16 sub-key bytes. The guess chain is accepted if all key bytes of the chain are among the top m candidates in their corresponding list ξ_i. The guess chain is rejected if at least one key byte of the chain falls outside the m top candidates in its corresponding list ξ_i.

2.3 Fault Tolerant Chain

In order to recover the key efficiently, the attacker usually hopes that a key chain includes 15 steps as introduced in [3]. For a chain, one of the common cases is that there are several steps in the path from the free variable to the end. If an error takes place in one of these steps, the key bytes computed in the following steps will be wrong in the key-recovery stage, which will result in the failure of the whole attack. Unfortunately, this kind of errors happen with non-negligible probability, which lead to low efficiency of Bogdanov's attack.

Wang et al. constructed a new chain named fault-tolerant chain (FTC)[27]. In their scheme, $k_i (i \geq 2)$ only depends on k_1 instead of any other 14 key bytes. There are 15 paths from k_1 to k_i ($i = 2, \cdots, 16$). If k_i is wrong (under the threshold line), they can still attempt to recover other key bytes. In their paper, the threshold of collision attack $Thr_{\varDelta}$ is set to 1. So, only Thr_k is taken into consideration in their scheme. Enlarging the threshold will lead to very complex key recovery. If k_i is under the threshold, their deduces that the chain is wrong. Subsequently, a practical exhaust search is performed to find the correct key.

Suppose that k_2 is wrong, and $\Delta_{(k_1,k_2)}$ is wrong, too. If $k_1 = \Delta_{(k_1,k_2)} \oplus k_2$ is still satisfied, then the attacker gets a wrong key byte value of k_2. However, he is completely unaware of the mistake. Actually, the threshold Thr_Δ is always set to 1. If Thr_Δ or (or and) Thr_k are set largely, the probability of this type of error is large. With the increase of Thr_Δ or (or and) Thr_k, experimental failures caused by this type of error increase significantly.

2.4 Group Verification Chain

Changhai et al. introduced group verification chain in [18]. Both frequency and weight based group verification chain are given, which can be used to reorder the key sequences under the condition that Thr_k and Thr_Δ are set largely. Group verification chain here is defined as the mutual verification among key bytes. Let ξ_i^k and $\xi_{\gamma+1}^t$ denote the k-th and t-th key values in ranks ξ_i and $\xi_{\gamma+1}$. $\Delta^m_{(k_i,k_{\gamma+1})}$ denotes the m-th value in the rank $\Delta_{(k_i,k_{\gamma+1})}$. Then,

$$\xi_i^k \oplus \xi_{\gamma+1}^t = \Delta^m_{(k_i,k_{\gamma+1})} \tag{4}$$

is satisfied if ξ_i^k, $\xi_{\gamma+1}^t$ and $\Delta^m_{(k_i,k_{\gamma+1})}$ are the correct ones. Then the frequencies of ξ_i^k and $\xi_{\gamma+1}^t$ are increased by 1. 120 sequences of $\Delta_{(k_a,k_b)}$ between any two key bytes k_a and k_b $(1 \le a < b \le 16)$ are also calculated. The correct key byte values are effectively supported that the Equation 4 is satisfied for most key byte values and Δs. Finally, the attacker gets the correct key.

3 Group Collision Attack

Suppose that the attacker obtains 16 key candidate sequences $\{\xi_i | i = 1, 2, \cdots, 16\}$ output by CPA and 120 Δ sequences output by CCA to construct $C^2_{(a,b)}$ chains for two key bytes k_a and k_b. We define a set $C^2_{a,b}$ including all chains (k_a, k_b) as

$$C^2_{a,b} = \{(k_a, k_b) | k_a \in K_a, k_b \in K_b\}, \tag{5}$$

each C^2 chain includes two sub key bytes k_a and k_b. K_a and K_b are the candidate space of k_a and k_b. k_a and k_b are within the threshold Thr_k, and $\Delta_{(k_a,k_b)}$ is within the threshold Thr_Δ. Each chain in the set $C^2_{a,b}$ satisfies that $\Delta_{(k_a,k_b)} = k_a \oplus k_b$. In this paper, we divide the 16 bytes entire key of AES into several groups (pieces). Group Collision Attack (GCA) is defined as collisions within each group. Each group establishes the connection among key bytes by multi-pairs of collisions.

We take a real experiment on power trace set downloaded from DPA *contest v*4 [1] implementing RSM [17] protected AES-256 for example to illustrate the attack efficiency of our KGE schemes. Other experimental results are shown in Section 5. Both Thr_k and Thr_Δ here are set to 10. The output of FGV-MDCA proposed by Ou et al [18] for each of the $1^{st} \sim 16^{th}$ guessing key byte are sorted in descending order. We divided the 16 bytes sub-key of AES-256 into 4 groups, each of which includes 4 key bytes. Since the first step of our KGE is deleting impossible key byte combinations within each group, here we take group (k_5, k_6, k_7, k_8) for example. Then, the key byte candidates and guessing Δs within the thresholds Thr_k and Thr_Δ are shown in **Table 1** and **Table 2**. Other key byte candidates and Δs listed here are used in Section 4.

Table 1. Candidates of the $3^{rd} \sim 14^{th}$ key bytes within Thr_k.

ξ_3	ξ_4	ξ_5	ξ_6	ξ_7	ξ_8	ξ_9	ξ_{10}	ξ_{11}	ξ_{12}	ξ_{13}	ξ_{14}
198	127	40	125	8	61	235	135	102	240	115	139
122	172	230	80	46	118	7	23	41	64	47	38
69	105	251	174	187	123	243	250	194	4	241	141
98	204	97	25	196	146	94	145	147	239	53	96
124	111	103	173	30	57	244	53	47	33	227	207
24	146	154	209	76	153	6	89	183	76	122	17
150	168	5	52	103	170	37	22	190	84	153	41
153	219	12	59	106	27	43	67	2	112	217	115
21	57	16	101	7	121	44	70	65	165	43	1
22	71	55	106	12	126	188	87	141	40	54	55

Table 2. Guessing Δs between some two of the $3^{rd} \sim 9^{th}$ key bytes within Thr_Δ.

$\Delta_{(k_3,k_4)}$	$\Delta_{(k_3,k_5)}$	$\Delta_{(k_4,k_5)}$	$\Delta_{(k_4,k_6)}$	$\Delta_{(k_5,k_6)}$	$\Delta_{(k_5,k_7)}$	$\Delta_{(k_6,k_7)}$	$\Delta_{(k_6,k_8)}$	$\Delta_{(k_7,k_8)}$	$\Delta_{(k_7,k_9)}$
185	238	87	2	49	32	117	64	53	162
148	244	134	102	135	236	169	11	115	83
153	110	77	208	189	252	130	6	126	97
115	223	129	136	138	70	83	9	124	180
174	168	17	174	85	91	23	116	191	249
20	163	231	123	193	117	32	239	140	10
180	197	242	234	209	215	185	26	164	14
128	42	147	150	141	96	102	140	113	251
110	32	203	80	112	66	0	212	154	227
159	100	161	3	75	125	53	209	194	185

We search key pairs (k_a, k_b) $(a, b \in [5, 8], a < b)$ of some two key bytes in Table 1 satisfying that their corresponding $\Delta_{(k_a,k_b)}$ are in Table 2. Each pair (k_a, k_b) constitutes a chain in $C^2_{a,b}$. Let $N_{C^2_{a,b}}$ denote the corresponding number of $C^2_{a,b}$ chains. As shown in Table 1 and Table 2, if k_5, k_6 and $\Delta_{(k_5,k_6)}$ are all within the thresholds, we then add k_5, k_6 to set $C^2_{5,6}$ (as shown in **Table 3**). For example, the value 40 of k_5, the value 125 of k_6, and the value 85 of $\Delta_{(k_5,k_6)}$ satisfy $85 = 40 \oplus 125$, so we add (40,125) to Table 3. Although the value 40 of k_5 and the value 80 of k_6 are within the threshold Thr_k, $40 \oplus 80 = 120$ is not in the threshold $\Delta_{(k_5,k_6)}$. So, key pair (40,80) is discarded. Finally, all key chains (k_5, k_6) are saved in the first two columns of Table 3. Each row saves a key chain so that the number of possible key chains in $C^2_{k_5,k_6}$ is 11. The attacker continues to find key chain sets $C^2_{k_5,k_7}$,$C^2_{k_6,k_7}$,$C^2_{k_6,k_8}$ and $C^2_{k_7,k_8}$. All C^2 chains calculated from Table 1 and Table 2 are shown in **Table 3**.

Table 3. C^2 chains of some two of the $5^{th} \sim 8^{th}$ key bytes.

$C^2_{5,6}$		$C^2_{5,7}$		$C^2_{6,7}$		$C^2_{6,8}$		$C^2_{7,8}$	
40	25	40	8	125	8	125	61	8	61
40	125	40	196	125	46	125	118	8	123
230	173	40	106	125	106	125	123	8	118
251	174	251	7	125	196	125	146	8	121
97	80	103	7	174	7	173	121	8	146
97	52			52	103	52	61	46	27
154	209			106	12	59	123	187	121
5	52			106	106	59	61	196	123
5	80							30	146
16	173							76	121
16	209							76	61
								103	27
								106	27
								7	121
								7	123
								7	118
								12	57

As shown in Table 3, there are only 11, 5, 8 and 17 possible combinations (chains) in set $C^2_{5,6}$, $C^2_{5,7}$, $C^2_{6,7}$, $C^2_{6,8}$ and $C^2_{7,8}$ respectively. If brute-force is used, 100 combinations between any two key bytes should be enumerated. So, the key candidate space becomes much smaller after deleting impossible key combinations in each group and recombination. Although the overhead of construction of C^2 chains of any two key bytes is $(Thr_k)^2 * (Thr_\Delta)$, if the AES full-key is divided into 8 independent groups, the complexity of calculating all C^2 chains is only $8 * (Thr_k)^2 * (Thr_\Delta)$. Similarly, the KGE schemes introduced in Sections 3.1 and 3.2 have even much smaller complexity of impossible key byte combinations deletion. Then, the attacker can search the key more efficiently in a much smaller key candidate space.

3.1 Key Chain Based Group Collision Attack

Actually, C^2 chains are the simplest side channel collision between any two key bytes. The attacker or evaluator can find several collisions among 3 or 4 key bytes simultaneously. For example, C^3 chain includes two collisions between k_a and k_b and between k_b and k_c. The three S-boxes correspond to three key bytes k_a, k_b and k_c. Then, a set including all C^3 chains of k_a, k_b and k_c is defined as

$$C^3_{a,b,c} = \{(k_a, k_b, k_c) | (k_a, k_b) \in C^2_{a,b}, (k_b, k_c) \in C^2_{b,c}\}, \tag{6}$$

which means both (k_a, k_b) and (k_b, k_c) are C^2 chains. Specifically, k_a, k_b and k_c are within the Thr_k; $\Delta_{(k_a,k_b)}$ and $\Delta_{(k_b,k_c)}$ are within the Thr_Δ; k_b in (k_a, k_b) and (k_b, k_c) are the same. The attacker or evaluator can construct more complex key chain based group collision attack schemes like C^4 by the same means as C^3 chains, which can be defined as

$$C^4_{a,b,c,d} = \{(k_a, k_b, k_c, k_d) | (k_a, k_b, k_c) \in C^3_{a,b,c}, (k_b, k_c, k_d) \in C^3_{b,c,d}\}. \tag{7}$$

Compared to $C^3_{a,b,c}$ and $C^3_{b,c,d}$, $C^4_{a,b,c,d}$ has higher requirements on candidates. For many C^3 chains, (k_a, k_b, k_c) and (k_b, k_c, k_d) are not established simultaneously. Since k_b is verified by k_a and k_c, and k_c is verified by k_b and k_d, the number of possible combinations in $C^4_{a,b,c,d}$ is much smaller than $N_{C^3_{a,b,c}} * N_{C^3_{b,c,d}}$. If the thresholds Thr_Δ and Thr_k are reasonable, the correct key bytes can be successfully used to construct C^4 chains and a lot of error C^3 chains are deleted.

Compared to C^2 chains, C^3 chains delete impossible combinations where k_b in (k_a, k_b) and (k_b, k_c) are not the same. So, N_e is reduced significantly. Taking a C^3 chain set $C^3_{5,6,7}$ for example, (k_5, k_6) and (k_6, k_7) are satisfied simultaneously. For example, $(k_5, k_6) = (40, 125)$ and $(k_6, k_7) = (125, 8)$ constitute $(k_5, k_6, k_7) = (40, 125, 8)$. However, $(k_5, k_6) = (16, 209)$ and $(k_6, k_7) = (125, 8)$ can not constitute $(k_5, k_6, k_7) = (16, 209, 8)$ or $(k_5, k_6, k_7) = (16, 125, 8)$. $N_{C^2_{5,6}}$ and $N_{C^2_{6,7}}$ are 11 and 8 respectively. To recover key pair (k_5, k_6, k_7), 88 key pairs should be enumerated. If $C^3_{5,6,7}$ chains are constructed, there are only 7 key pairs should be taken into consideration. The key search space is reduced obviously.

The construction of C^3 chains using guessing key bytes in Table 1 and Δs in Table 2 are shown in **Table 4**. We get a conclusion that $N_{C^3_{5,6,7}} = 7$ and $N_{C^3_{6,7,8}} = 14$. Compared to C^3 chains, C^4 chains are more efficient. There are only 13 C^4 chains in thresholds (as shown in **Table 7**). The constraints of $C^4_{a,b,c,d}$ are more strict than $C^3_{a,b,c}$ and $C^3_{b,c,d}$. Compared with 10^4 of exhausting key bytes in the thresholds, only 13 possible combinations of k_5, k_6, k_7 and k_8 are enumerated by the attacker in the second round.

The advantage of key chain based GCA schemes is that they are simple to construct and suitable for small Thr_k and Thr_Δ. Since more strict constraint may delete the correct key bytes. However, the number of error C^2 chains deleted by this method is still very limited. After all, the key search space ${Thr_k}^{16}$ is too huge. For example, if Thr_k is set to 32, then the key candidate space is 2^{80}.

Actually, for key chain based GCA schemes, key bytes in the middle positions are verified two times, and the two key bytes at two ends are verified only once. So, key bytes in the middle positions are more credible and more likely to be the correct ones. For example, Since k_b and k_c in $C^4_{a,b,c,d}$ can be verified by two collisions (k_a, k_b), (k_b, k_c) and (k_b, k_c), (k_c, k_d) separately. However, k_a and k_d are not verified as strongly as k_b and k_c. Thus, k_b and k_c are more credible and are more likely to be the correct keys. Each verification means that more impossible key combinations are removed, the attacker or evaluator can get smaller key candidate space after recombination.

In order to improve the reliability of key bytes located in two ends of key chain based GCA schemes and further reduce the key search space, we propose ring based GCA schemes in the next subsection.

3.2 Key Ring Based Group Collision Attack

In Section 3.1, we introduce our key chain based group collision attack schemes under small thresholds Thr_k and Thr_Δ. If the thresholds are small, the correct key bytes may fall outside of them. If the attacker enlarges thresholds, the success

Table 4. C^3 chains of the $5^{th} \sim 8^{th}$ key bytes.

$C^3_{5,6,7}$			$C^3_{6,7,8}$		
40	125	8	125	8	61
40	125	46	125	8	123
40	125	106	125	8	118
40	125	196	125	8	121
251	174	7	125	8	146
97	52	103	125	46	27
5	52	103	125	106	27
			125	196	123
			174	7	121
			174	7	123
			174	7	118
			52	103	27
			106	12	57
			106	106	27

Table 5. R^3 rings of the $5^{th} \sim 8^{th}$ key bytes.

$R^3_{5,6,7}$			$R^3_{6,7,8}$		
40	125	8	125	8	61
40	125	196	125	8	118
40	125	106	125	8	123
251	174	7	125	8	146
			125	196	123

rate [24] will be improved. However, it is very time-consuming, since the key candidate space becomes larger. In order to improve the reliability of key bytes located in two ends of chain and further reduce the key candidate space efficiently in large thresholds, we propose the concept of the **Key Ring** in this section. A key ring R^n consists of n C^2 chains (k_1, k_2), (k_2, k_3),... , (k_{n-1}, k_n) and (k_1, k_n). A set of ring $R^3_{a,b,c}$ constituting of k_a, k_b and k_c is defined as

$$R^3_{a,b,c} = \{(k_a, k_b, k_c)|(k_a, k_b) \in C^2_{a,b}, (k_b, k_c) \in C^2_{b,c}, (k_a, k_c) \in C^2_{a,c}\}, \quad (8)$$

which means (k_a, k_b), (k_b, k_c) and (k_a, k_c) are C^2 chains simultaneously. So, to construct the ring R^3, the attacker only needs to traverse the C^2 table twice (see Equation 8), its complexity is similar to the construction of the C^4 chains.

The $R^3_{5,6,7}$ and $R^3_{6,7,8}$ rings constructed by the corresponding guessing key bytes and guessing Δs in Table 1 and Table 2 are shown in **Table 5**. We get a conclusion that $N_{R^3_{5,6,7}} = 4$ and $N_{R^3_{6,7,8}} = 5$. However, we get another conclusion from Table 4 that $N_{C^3_{5,6,7}} = 7$ and $N_{C^3_{6,7,8}} = 14$. This indicates that, R^3 scheme, which add a constraint (a pair of collision) on C^3 chains, can effectively reduce the key candidate space. This also indicates that ring based GCA schemes are more efficient than chain based GCA schemes when deleting impossible key bytes combinations within group, due to more strict constraints (collisions).

The $R^3_{a,b,c}$ only has a more pair of collision (k_a, k_c) than $C^3_{a,b,c}$. However, a ring is constructed since the existence of this pair of collision. Each of the three key bytes k_a,k_b,k_c on the ring $R^3_{a,b,c}$ is verified by the other two key bytes. Thus, the probability of these three key bytes being the correct ones increases. Key ring based GCA can delete more impossible combinations than key chain based GCA. The attacker can also construct intersecting rings of two R^3 rings. A $R^{2-3}_{a,b,c;a,b,d}$ including 2 rings $R^3_{a,b,c}$, $R^3_{a,b,d}$, which has more stringent constraint than R^3 rings. A set $R^{2-3}_{a,b,c;a,b,d}$ is defined as

$$R^{2-3}_{b,c,a;b,c,d} = \{(k_a, k_b, k_c, k_d)|(k_a, k_b, k_c) \in R^3_{a,b,c}, (k_b, k_c, k_d) \in R^3_{b,c,d}\}, \quad (9)$$

where $a < b < c < d$. Actually, R^{2-3} rings are easy to construct, the attacker can traverse the R^3 table twice (see Equation 9), search the sets $R^3_{a,b,c}$ and $R^3_{b,c,d}$, and select all double-rings from each of them if k_b, k_c are equal.

The number of R^{2-3} rings $N_{R^{2-3}_{6,7,5;6,7,8}}$ constructed by the guessing key bytes and guessing Δs in Table 1 and Table 2 is 5 (as shown in **Table 8**). However, if the attacker exhausts all the 4 key bytes in the threshold $Thr_k = 10$, the complexity is 10^4. Obviously, the attacker gets higher efficient than C^3, C^4 and R^3 schemes. The complexity of $R^{2-3}_{b,c,a;b,c,d}$ rings construction is average $N_{R^3_{5,6,7}} \cdot N_{R^3_{6,7,8}}$ more complex than that of R^3 schemes.

In the case of large thresholds Thr_k and Thr_Δ, the correct key bytes and Δs are within thresholds no matter what kind of constraints we use. The more harsh conditions, the smaller number of remaining R^{2-3} rings. Moreover, we divide the 16-byte subkey of the AES algorithm into 4 groups, which are independent of each other. The number of remaining key rings in each group is small, the attacker can efficiently exhaust the 16-byte key in a much smaller candidate space. The attacker can also construct more complex rings including more collisions. With the increase of Thr_k and Thr_Δ, the number of R^3 or C^3 increases very fast, more complex rings or chains mean more loops in the program. so the attacker also needs to consider the efficiency of program.

4 Key Grouping Enumeration

In Section 3, we introduce our GCA, which is a distinguisher to select possible key combinations and delete impossible key combinations in each group. This is the first step of our KGE. The second step of KGE is impossible key combinations deletion among groups, which we will introduce in this section. The third (last) step is key enumeration, the attacker can use the state-of-art key enumeration solutions to search the key. For simplicity, here we only use key exhaustion.

The correct key bytes and Δs are within thresholds if Thr_k and Thr_Δ are set largely enough. In this case, each correct chain or ring is within the threshold. What the attacker needs to do is using GCA proposed in Section 3 to delete impossible combinations in each big group. Suppose that he divides the entire key into 4 groups, which are independent of each other, and uses C^4 chains to delete impossible key combinations in each group. Here we recombine the remaining key candidates of the first step.

Actually, the possible key candidate combinations in each group have been greatly reduced after the first round of deletion. Suppose that there are n_1, n_2, n_3 and n_4 chains in $C^4_{1,2,3,4}$, $C^4_{5,6,7,8}$, $C^4_{9,10,11,12}$ and $C^4_{13,13,15,16}$ respectively, then a very simple solution to re-combine the entire key is calculate every possible combinations. By doing this, the attacker will get $n_1 \cdot n_2 \cdot n_3 \cdot n_4$ possible combinations. This value increases very fast with Thr_k and Thr_Δ. Obviously, this is not a good combination strategy. So, here we propose a new solution to re-combine the entire key in KGE. We use verification chain to re-combine the remaining tuples in each group and carry out a second round impossible key bytes combinations deletion.

Let $C^4_{a,b,c,d}$ and $C^4_{e,f,g,h}$ denote two verification chain sets to verify chains in set $C^4_{c,d,e,f}$. If there has more than one chain in set $C^4_{c,d,e,f}$ satisfying that k_c, k_d in $C^4_{a,b,c,d}$ and k_e, k_f in $C^4_{e,f,g,h}$. Then, we define a new verified C^4 chain set as

$$V^4_{c,d,e,f} = \{(k_c,k_d,k_e,k_f)|(k_a,k_b,k_c,k_d) \in C^4_{a,b,c,d}, (k_e,k_f,k_g,k_h) \in C^4_{e,f,g,h}\}. \tag{10}$$

Actually, verification chains are well used to delete impossible entire key combinations. The verified set $V^4_{c,d,e,f}$ is a sub set of $C^4_{c,d,e,f}$.

For example, we use $C^4_{3,4,5,6}$ and $C^4_{7,8,9,10}$ to verify $C^4_{5,6,7,8}$, and use $C^4_{7,8,9,10}$ and $C^4_{11,12,13,14}$ to verify $C^4_{9,10,11,12}$ (as shown in Table 7). Δ values are shown in Table 2 and Table 6. We delete chains (k_5,k_6,k_7,k_8) in set $C^4_{5,6,7,8}$ if (k_3,k_4,k_5,k_6) are not in $C^4_{3,4,5,6}$, and delete (k_5,k_6,k_7,k_8) if (k_7,k_8,k_9,k_{10}) are not in $C^4_{7,8,9,10}$. $C^4_{9,10,11,12}$ are processed in the same way. There are 13, 16 possible combinations in the set $C^4_{5,6,7,8}$ and $C^4_{9,10,11,12}$ respectively before verification. However, 4 and 3 chains are left after verification (as shown in **Table 9**). So, the number of possible combinations drops from $13 \cdot 16 = 208$ to $3 \cdot 4 = 12$, which indicates the high efficiency of our KGE.

Table 6. Guessing Δs between some two of the $8^{th} \sim 14^{th}$ key candidates.

$\Delta_{(k_8,k_9)}$	$\Delta_{(k_8,k_{10})}$	$\Delta_{(k_9,k_{10})}$	$\Delta_{(k_9,k_{11})}$	$\Delta_{(k_{10},k_{11})}$	$\Delta_{(k_{10},k_{12})}$
214	186	108	196	225	119
59	199	109	133	168	171
190	172	252	141	174	199
204	150	235	78	233	247
94	239	70	120	195	55
148	16	122	194	238	27
224	144	150	170	20	18
112	42	199	249	223	31
241	43	17	198	149	202
129	123	226	224	57	111
$\Delta_{(k_{11},k_{12})}$	$\Delta_{(k_{11},k_{13})}$	$\Delta_{(k_{12},k_{13})}$	$\Delta_{(k_{12},k_{14})}$	$\Delta_{(k_{13},k_{14})}$	
150	21	131	123	49	
78	191	197	50	197	
98	151	1	47	248	
142	255	128	223	254	
38	83	41	214	98	
170	2	188	219	136	
219	28	214	125	120	
139	6	223	121	62	
133	152	95	153	172	
246	136	105	89	85	

In order to enhance the verification, we also verify R^{2-3} in this way. $R^{2-3}_{4,5,3;4,5,6}$, $R^{2-3}_{6,7,5;6,7,8}$, $R^{2-3}_{8,9,7;8,9,10}$, $R^{2-3}_{10,11,9;10,11,12}$ and $R^{2-3}_{12,13,11;12,13,14}$ calculated from Table 1, 2 and 6 are shown in **Table 8**. We also use $R^{2-3}_{4,5,3;4,5,6}$, $R^{2-3}_{8,9,7;8,9,10}$ and $R^{2-3}_{12,13,11;12,13,14}$ to verify $R^{2-3}_{6,7,5;6,7,8}$ and $R^{2-3}_{10,11,9;10,11,12}$. Finally, there are only a possible R^{2-3} rings $(40,125,8,61)$, $(235,135,102,240)$ for these two verified R^{2-3} rings respectively (as shown in **Table 10**). The attacker only has to enumerate $(40,125,8,61,235,135,102,240)$ for key bytes $(k_5,k_6,k_7,k_8,k_9,k_{10},k_{11},k_{12})$. Actually, this only remaining combination corresponds to the correct key bytes

Table 7. C^4 chains of the $3^{rd} \sim 14^{th}$ key bytes.

$C^4_{3,4,5,6}$				$C^4_{5,6,7,8}$				$C^4_{7,8,9,10}$				$C^4_{9,10,11,12}$				$C^4_{11,12,13,14}$			
198	127	40	25	40	125	8	61	8	61	235	135	235	135	102	240	102	240	115	139
198	127	40	125	40	125	8	123	8	61	235	23	235	135	102	40	102	240	115	141
98	219	16	173	40	125	8	118	8	61	235	145	235	135	102	4	102	240	115	17
98	219	16	209	40	125	8	121	8	61	235	250	235	135	102	64	102	240	115	38
24	172	251	174	40	125	8	146	8	61	6	250	235	135	190	40	102	240	53	96
21	172	251	174	40	125	46	27	8	61	6	23	235	135	190	240	102	240	241	207
				40	125	106	27	8	61	188	87	7	22	190	40	102	240	47	17
				40	125	196	123	8	61	188	250	7	22	190	240	102	240	153	17
				251	174	7	121	8	118	6	250	7	22	2	76	190	240	115	139
				251	174	7	123	8	118	6	23	244	22	190	40	190	240	115	141
				251	174	7	118	8	146	6	250	244	22	190	240	190	240	115	17
				97	52	103	27	8	146	6	23	244	22	2	76	190	240	115	38
				5	52	103	27	30	146	6	250	188	87	190	40	190	240	53	96
								30	146	6	23	188	87	190	240	190	240	241	207
								76	61	235	135	188	87	194	84	190	240	47	17
								76	61	235	23	188	87	194	76	190	240	153	17
								76	61	235	145								
								76	61	235	250								
								76	61	6	250								
								76	61	6	23								
								76	61	188	87								
								76	61	188	250								
								7	118	6	250								
								7	118	6	23								

Table 8. Some R^{2-3} rings of the $3^{rd} \sim 14^{th}$ key bytes.

$R^{2-3}_{4,5,3;4,5,6}$				$R^{2-3}_{6,7,5;6,7,8}$				$R^{2-3}_{8,9,7;8,9,10}$				$R^{2-3}_{10,11,9;10,11,12}$				$R^{2-3}_{12,13,11;12,13,14}$			
198	127	40	125	40	125	8	61	8	61	188	250	235	135	102	240	102	240	115	139
198	127	40	25	40	125	8	118	8	61	6	250	235	135	102	64	102	240	115	38
21	172	251	174	40	125	8	123	8	61	6	23					102	240	115	141
				40	125	8	146	8	61	235	135								
				40	125	196	123	8	61	235	250								
								8	61	235	145								
								8	61	235	23								

Table 9. C^4 chains of the $5^{th} \sim 8^{th}$ key bytes.

$C^4_{5,6,7,8}$				$C^4_{9,10,11,12}$			
40	125	8	61	188	87	190	240
40	125	8	118	235	135	102	240
40	125	8	146	235	135	190	240
251	174	7	118				

Table 10. Verified $R^{2-3}_{6,7,5;6,7,8}$ rings of the $5^{th} \sim 8^{th}$ key bytes.

$R^{2-3}_{6,7,5;6,7,8}$				$R^{2-3}_{10,11,9;10,11,12}$			
40	125	8	61	235	135	102	240

$k_5 \sim k_{12}$. So, compared to C^4 chains, R^{2-3} rings are more powerful. Similarly, we also use $R^{2-3}_{12,13,11;12,13,14}$ and $R^{2-3}_{16,1,15;16,1,2}$ to verify $R^{2-3}_{14,15,13;14,15,16}$, and use $R^{2-3}_{16,1,15;16,1,2}$ and $R^{2-3}_{4,5,3;4,5,6}$ to verify $R^{2-3}_{2,3,1;2,3,4}$.

Moreover, the attacker or evaluator can also put forward different levels of requirements for verification according to the size of thresholds Thr_k and Thr_{Δ}. Larger thresholds may need higher level of requirements. What the attacker should take into consideration is that the combination of smaller groups may bring in greater computation. Therefore, we recommend that the entire key should not be divided into very smaller groups(pieces). For example, dividing the entire key of AES-256 into $3 \sim 5$ groups may be a good decision.

5 Experimental Results

Our experiments are performed on an RSM [17] protected AES algorithm implemented on the Side-channel Attack Standard Evaluation Board (SASEBO). We use 40000 power trace set downloaded from the website of DPA *contest v*4 [1]. We then implement our experiments on MATLAB R2014a on a desktop computer with 4 Intel Core i7-3770 CPUs, 4 GB RAM and 500 GB memory.

We find the time samples from 100001 to 101000 of the first S-box within the first round by using the optimal power consumption model of RSM [17] protected AES proposed by Moradi et al [15]. The time samples of the other 15 S-boxes are aligned to these of the first S-box. Then, we perform CCA combined with TA [16] on 4000 power traces to extract 4 interesting points from time interval of about a clock cycle suggested in [22]. Like Ou et al. in [18], we also use group verification chain to reorder the outputs of CPA so that the average positions of correct key bytes are closer to the top of sequences. Another advantage of group verification chain is, when the threshold Thr_k is reasonable, the possibility that the correct key bytes fall outside the threshold is reduced. In fact, our algorithm does not take up too much memory space. In order to quickly getting the experimental results, we run 4 MATLAB main programs simultaneously on our desktop computer. Each main program takes up less than 500MB memory.

5.1 Experimental Results Under Different Thresholds Thr_Δ

In this subsection, we compare our KGE scheme with TC proposed by [3] and FTC proposed by Wang et al [27]. The search complexity of FTC is related to the number of wrong key bytes as given by Wang et al [27]. They indicated that the maximum number of candidates is $2^8 \cdot (2^8)^n \cdot \binom{15}{n}$ for n wrong key bytes. If there are total t times that more than one candidates are in the threshold, then $15 - t$ errors can be detected. Then the number of key candidates the attacker can only need to enumerate is almost

$$2^8 \cdot \left(2^8\right)^n \cdot \binom{t}{t+n-15}. \tag{11}$$

Each experiment below is repeated 200 times. When average 100 power traces are used and the Thr_k is set to 16 (the candidate space here is 2^{64}), if the Thr_Δ is from 1 to 20, the time consumption and success rate of the 4 schemes TC, FTC, C^4, and R^{2-3} are shown in **Fig.2**. Since Thr_Δ of TC and FTC is set to 1 in [3] and [27], and no practical schemes for larger Thr_Δ are given in these two papers. So, here we still set Thr_Δ of TC and FTC to 1. The time consumption of TC and FTC increases very slow when Thr_Δ is from 2 to 20. It is still less than 0.025 seconds when $Thr_\Delta = 20$. Compared to TC and FTC, our C^4 and R^{2-3} consume more time, nearly 0.4 second is needed when $Thr_\Delta = 20$. This value increases fast if both Thr_k and Thr_Δ increase. For example, if $Thr_k = 32$ and $Thr_\Delta = 26$, it may need several minutes for these two schemes.

The success rate of the 4 schemes TC, FTC, C^4, and R^{2-3} is very low under small threshold Thr_Δ. For example, if $Thr_\Delta = 2$, the success rate of these 4

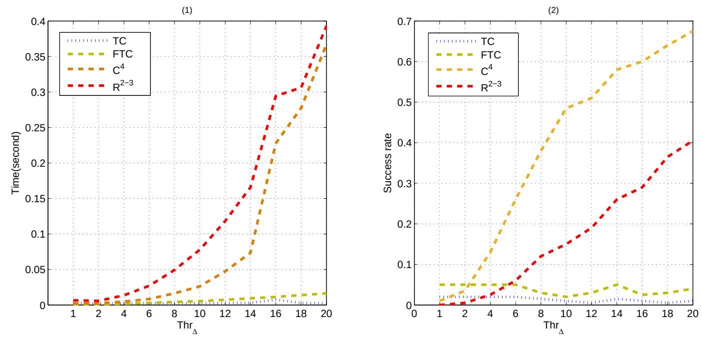

Fig. 2. Time consumption and success rate of 4 schemes under different Thr_Δ.

schemes is about 0.02, 0.05, 0.035, 0.05 respectively. With the increase of Thr_Δ, more correct Δs fall within the thresholds, the success rate increases. When Thr_Δ reaches 20, the success rate of these 4 schemes is about 0.01, 0.04, 0.675 and 0.405. Since some correct Δs fall out of Thr_Δ, the success rate of C^4 is higher than that of R^{2-3}.

Let N_e denote the number of possible key candidates to be enumerated. KGE, the new strategy proposed in this paper is aimed at recover the key more efficiently. When Thr_Δ is from 1 to 20, N_e is shown in **Fig.3**. With the increase of Thr_Δ, N_e of C^4 scheme grow very fast. When Thr_Δ is from 2 to 20, the attacker has to enumerate $2^0 \sim 2^6$ and $2^{17} \sim 2^{30}$ key candidates respectively. Larger thresholds mean larger key candidate space.

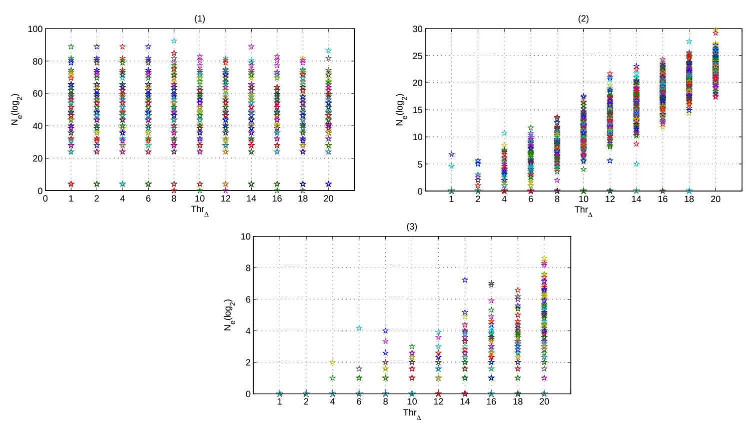

Fig. 3. Different N_e of 3 schemes when Thr_Δ is from 2 to 20.

However, R^{2-3} with more strict constraints requires fewer key candidates to be enumerated. When Thr_Δ reaches 20, R^{2-3} scheme only needs to enumerate almost 390 key candidates. Compared to C^4 and R^{2-3} solutions, FTC appears to be random since Thr_Δ is set to 1, $2^{20} \sim 2^{90}$ key candidates may need to enumerate, which may far beyond exhaustion. N_e of TC is usually very small (i.e. 1), which we don't give in Fig.3, Fig.5 and Fig.7.

5.2 Experimental Results Under Different Thresholds Thr_k

When average 100 power traces are used and the Thr_Δ of C^4 and R^{2-3} is set to 14 (Thr_Δ of TC and FTC is always set to 1), if the Thr_k is set to from 2 to 22 (the key candidate space is $2^{16} \sim 2^{72}$), the time consumption and success rate of the 4 schemes are shown in **Fig. 4**. Like time consumption in Fig.2, TC and FTC is very fast, changing Thr_k does not bring too much computation. When $Thr_\Delta = 14$ and $Thr_k = 16$, nearly 0.0483 and 0.0925 second is used. When $Thr_k = 14$ and $Thr_\Delta = 16$, nearly 0.0734 second and 0.1616 second is used. It seems that enlarging Thr_Δ will consume more time than enlarging Thr_k, more key candidates need to enumerate (as shown in Fig.3 and Fig.5).

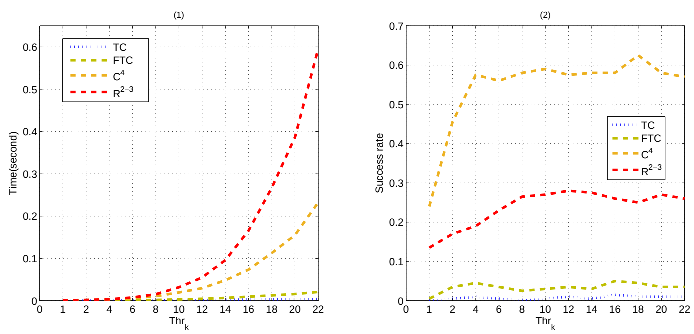

Fig. 4. Time consumption and success rate of 4 schemes under different Thr_k.

However, the success rate of TC and FTC is still very low. Increasing Thr_k does not significantly increase the success rate. The success rate of C^4 and R^{2-3} reach the highest when Thr_k is 4 and 8 respectively. It then remains stable (as shown in Fig.4). Continuing to increase the threshold does not result in higher success rate, while the attacker needs to enumerate more candidates (as shown in Fig.5). So, it will be better for the attacker to choose a reasonable Thr_k.

The complexity of the 3 schemes FTC, C^4 and R^{2-3} is very different. Like in Fig.3, N_e of FTC seems to be random, enlarging Thr_k does not reduce the number of key candidates. A large number of key candidates are left in many experiments, which is unreachable (i.e. larger than 2^{60}). Compared to Fig.3, enlarging Thr_k also brings computation. When $Thr_k = 2$, N_e of C^4 scheme is

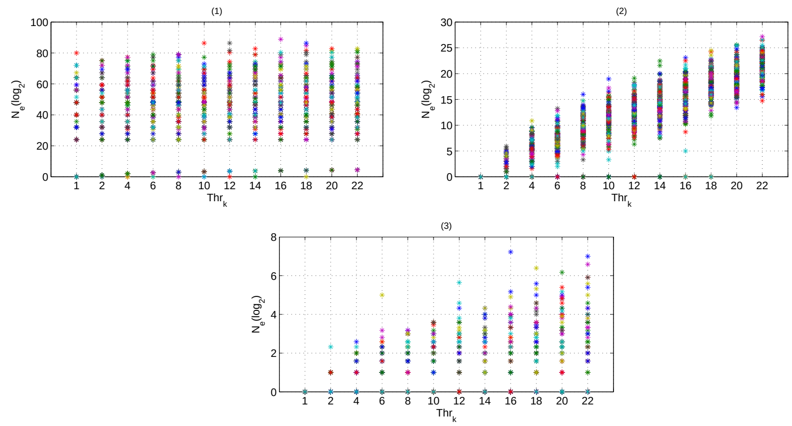

Fig. 5. Different N_e of 4 schemes when Thr_k is from 2 to 22.

$2^0 \sim 2^6$. When $Thr_k = 22$, this value becomes $2^{14} \sim 2^{26}$. However, our R^{2-3} scheme only needs to enumerate less than 2^7 possible candidates. The attacker can easily exhaust the correct entire key in the third step of KGE. If the attacker enlarges Thr_k, Thr_Δ, and the remaining key candidates of R^{2-3} is still very large. Then, the state-of-art key enumeration solutions can be used to search the entire key. In order to reduce N_e, the attacker can choose a more strictly constrained scheme under large thresholds.

5.3 Experimental Results Under Different Numbers of Power Traces

We also compare the time consumption and success rate under different numbers of power traces. Thr_k and Thr_Δ are set to 16 and 14 respectively. The experimental results are shown in **Fig.6**. Let N_p denote the number of power traces used. The time consumption of TC is almost the same when N_p is from 40 to 260 compared to $0.0066 \sim 0.0114$ second used in FTC. Compared to TC and FTC, our C^4 and R^{2-3} spend $0.0345 \sim 0.1534$ and $0.1014 \sim 0.2405$ second. This indicates that, with the increase of N_p, more correct key bytes and Δs are within Thr_k and Thr_Δ, more time are needed to construct chains and rings.

Compared to increase Thr_k or Thr_Δ, it will be more efficient to increase the number of power traces. The success rate of 4 schemes increases fast with the number of power traces used in each repetition (as shown in **Fig.6(2)**). When 100 power traces are used, the success rate of these 4 schemes are 0.0150, 0.0500,0.5800 and 0.2600 respectively. These 4 values soon become 0.0550, 0.1500, 0.8000 and 0.5200 when average 120 power traces are used. The success rate of these 4 schemes reaches 0.7850, 0.9650, 0.9850 and 0.9750 respectively when average 260 power traces are used. Actually, the success rate of our C^4 and R^{2-3}

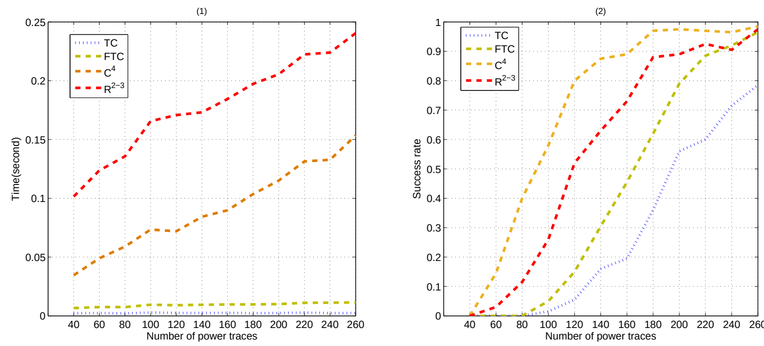

Fig. 6. Time consumption and success rate of 4 schemes under different numbers of power traces.

schemes are more higher than that of TC and FTC, which indicates that our C^4 and R^{2-3} schemes can significant improve the efficiency of TC and FTC.

The N_e of C^4 and R^{2-3} when different numbers of power traces being used are shown in **Fig.7**, which changes much smaller compared to different thresholds since the fixed Thr_k and Thr_Δ(as shown in **Fig.3 and Fig.5**). N_e of FTC decreases with increase of N_e. When average 40, 80, 120, 160, 200 power traces are used, the range of N_e is $2^{60} \sim 2^{120}$, $2^{30} \sim 2^{100}$, $2^{24} \sim 2^{80}$, $2^{24} \sim 2^{64}$ and $2^4 \sim 2^{40}$ respectively. This indicates that, with more power traces used in each repetition, more correct Thr_k and Thr_Δ locate in the front of sequences output by side channel distinguishers. The advantages of TC and FTC began to appear.

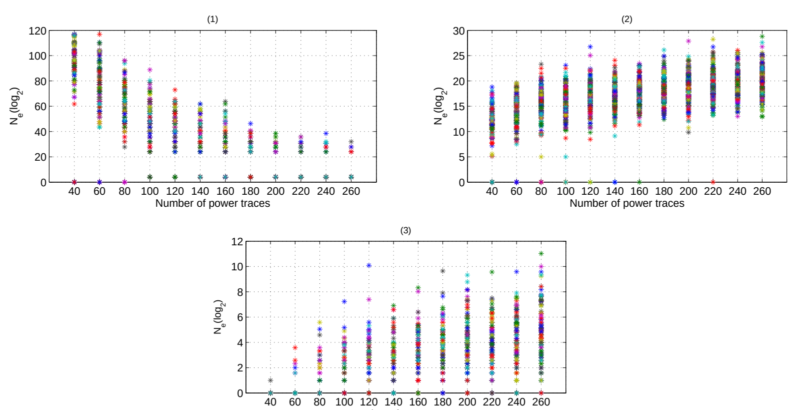

Fig. 7. Different N_e of 3 schemes under different numbers of power traces.

N_e of C^4 under different numbers of power traces also increases, but not as fast as it in Fig.3 and Fig.5. When average 40, 260 power traces are used, the attacker has to enumerate $2^5 \sim 2^{20}$ and $2^{12} \sim 2^{28}$ key candidates. If R^{2-3} is used, $2^0 \sim 2^1$ and $2^0 \sim 2^{11}$ key candidates are needed to enumerate. However, similar to N_e in Fig.3 and Fig.5, N_e of R^{2-3} is smaller than 2^8 in most of repetitions.

6 Thr_k and Thr_Δ in KGE

In the Section 5, we discuss the experimental results under different thresholds Thr_k, Thr_Δ, and different number of power traces. Actually, it's hard for attacker to determine the correlation between Thr_k and Thr_Δ in our KGE. If the attacker uses small thresholds and a lot of constraints, the correct key is easy to be deleted. So, he had better use simple constraints such as C^3 chains, C^4 chains, etc.

Actually, the locations of Thr_k and Thr_Δ are determined by the outputs of the distinguishers. For example, correlation-enhanced collision attack determines the locations of Thr_Δ, CPA determines the locations of correct sub keys candidates. If the thresholds Thr_k and Thr_Δ are large, the attacker has to enumerate large number of key candidates. The attacker can appropriately increase the constraints and construct more complex constraints such as R^3 rings, R^{2-3} rings to delete wrong key candidates. However, if the attacker's constraints are too complex, the complexity of algorithm itself is a big problem. Although the constraint conditions can effectively reduce the number of key rings or chains, it also increases the complexity of the construction of the them. Moreover, very reasonable thresholds Thr_k and Thr_Δ are very hard to find. So, the attacker or evaluator needs to introduce fault tolerance schemes into KGE to reduce the probability of accidentally deleting the correct key bytes.

7 Conclusions and Open Problems

Key enumeration solutions are post-processing schemes for the output sequences of side channel distinguishers. The attacker or evaluator enumerates the key candidates from the most possible one to the most impossible one, he may not obtain the entire key directly from the outputs of the distinguisher. Since the correct sub-key bytes are not always located at the top of the output sequences. Therefore, the attacker or evaluator needs to use certain algorithms to search the entire key. However, This kind of solutions are time-consuming, the attacker or evaluator has to spend several days or months to enumerate a key candidate space 2^{40}. The efficiency of these solutions is still very low.

In this paper, a new distinguisher named GCA is given. Moreover, a divide and conquer strategy named KGE is proposed. Key chain and key ring based KGE schemes are introduced in detail. Experiments results show that our KGE schemes can significantly reduce key candidate space, which can be regarded as a very powerful pre-processing tool of key enumeration.

There are still several open problems of KGE. The first open problem is fault tolerance of KGE to reduce the probability of accidentally deleting the correct subkey bytes. The second open problem is how to delete the impossible combinations under large thresholds Thr_k and Thr_Δ. Since compared to 2^{128}, $2^{64} \sim 2^{71}$ in this paper is still very small. With the increase of both two thresholds, more and more combinations meet the conditions. Recently, there were several papers discussing key enumeration in parallel [13, 21, 8]. Our KGE is also very easy to perform in parallel since the 16-byte key is divided into several independent groups. Each group can be calculated independently. The efficient KGE parallel algorithms are also a open problem.

References

1. Dpa contest. `http://www.dpacontest.org/home/`.
2. D. J. Bernstein, T. Lange, and C. van Vredendaal. Tighter, faster, simpler side-channel security evaluations beyond computing power. *IACR Cryptology ePrint Archive*, 2015:221, 2015.
3. A. Bogdanov and I. Kizhvatov. Beyond the limits of DPA: combined side-channel collision attacks. *IACR Cryptology ePrint Archive*, 2010:590, 2010.
4. A. Bogdanov, I. Kizhvatov, K. Manzoor, E. Tischhauser, and M. Witteman. Fast and memory-efficient key recovery in side-channel attacks. In *Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers*, pages 310–327, 2015.
5. A. Bogdanov, I. Kizhvatov, and A. Pyshkin. Algebraic methods in side-channel collision attacks and practical collision detection. In *Progress in Cryptology - INDOCRYPT 2008, 9th International Conference on Cryptology in India, Kharagpur, India, December 14-17, 2008. Proceedings*, pages 251–265, 2008.
6. E. Brier, C. Clavier, and F. Olivier. Correlation power analysis with a leakage model. In *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, pages 16–29, 2004.
7. S. Chari, J. R. Rao, and P. Rohatgi. Template attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, pages 13–28, 2002.
8. L. David and A. Wool. A bounded-space near-optimal key enumeration algorithm for multi-subkey side-channel attacks. In *Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings*, pages 311–327, 2017.
9. B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel. Mutual information analysis. In *Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings*, pages 426–442, 2008.
10. C. Glowacz, V. Grosso, R. Poussier, J. Schüth, and F. Standaert. Simpler and more efficient rank estimation for side-channel security assessment. In *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, pages 117–129, 2015.
11. P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pages 388–397, 1999.

12. D. P. Martin, L. Mather, E. Oswald, and M. Stam. Characterisation and estimation of the key rank distribution in the context of side channel evaluations. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, pages 548–572, 2016.
13. D. P. Martin, J. F. O'Connell, E. Oswald, and M. Stam. Counting keys in parallel after a side channel attack. In *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, pages 313–337, 2015.
14. A. Moradi. Statistical tools flavor side-channel collision attacks. In *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, pages 428–445, 2012.
15. A. Moradi, S. Guilley, and A. Heuser. Detecting hidden leakages. In *Applied Cryptography and Network Security - 12th International Conference, ACNS 2014, Lausanne, Switzerland, June 10-13, 2014. Proceedings*, pages 324–342, 2014.
16. A. Moradi, O. Mischke, and T. Eisenbarth. Correlation-enhanced power analysis collision attack. In *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, pages 125–139, 2010.
17. M. Nassar, Y. Souissi, S. Guilley, and J. Danger. RSM: A small and fast countermeasure for aes, secure against 1st and 2nd-order zero-offset scas. In *2012 Design, Automation & Test in Europe Conference & Exhibition, DATE 2012, Dresden, Germany, March 12-16, 2012*, pages 1173–1178, 2012.
18. C. Ou, Z. Wang, D. Sun, X. Zhou, and J. Ai. Group verification based multiple-differential collision attack. In *Information and Communications Security - 18th International Conference, ICICS 2016, Singapore, November 29 - December 2, 2016, Proceedings*, pages 145–156, 2016.
19. J. Pan, J. G. J. van Woudenberg, J. den Hartog, and M. F. Witteman. Improving DPA by peak distribution analysis. In *Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers*, pages 241–261, 2010.
20. R. Poussier, V. Grosso, and F. Standaert. Comparing approaches to rank estimation for side-channel security evaluations. In *Smart Card Research and Advanced Applications - 14th International Conference, CARDIS 2015, Bochum, Germany, November 4-6, 2015. Revised Selected Papers*, pages 125–142, 2015.
21. R. Poussier, F. Standaert, and V. Grosso. Simple key enumeration (and rank estimation) using histograms: An integrated approach. In *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, pages 61–81, 2016.
22. C. Rechberger and E. Oswald. Practical template attacks. In *Information Security Applications, 5th International Workshop, WISA 2004, Jeju Island, Korea, August 23-25, 2004, Revised Selected Papers*, pages 440–456, 2004.
23. K. Schramm, G. Leander, P. Felke, and C. Paar. A collision-attack on AES: combining side channel- and differential-attack. In *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, pages 163–175, 2004.
24. F. Standaert, T. Malkin, and M. Yung. A unified framework for the analysis of side-channel key recovery attacks. In *Advances in Cryptology - EUROCRYPT 2009,*

28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings, pages 443–461, 2009.
25. N. Veyrat-Charvillon, B. Gérard, M. Renauld, and F. Standaert. An optimal key enumeration algorithm and its application to side-channel attacks. In *Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers*, pages 390–406, 2012.
26. N. Veyrat-Charvillon, B. Gérard, and F. Standaert. Security evaluations beyond computing power. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 126–141, 2013.
27. D. Wang, A. Wang, and X. Zheng. Fault-tolerant linear collision attack: A combination with correlation power analysis. In *Information Security Practice and Experience - 10th International Conference, ISPEC 2014, Fuzhou, China, May 5-8, 2014. Proceedings*, pages 232–246, 2014.
28. X. Ye, T. Eisenbarth, and W. Martin. Bounded, yet sufficient? how to determine whether limited side channel information enables key recovery. In *Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers*, pages 215–232, 2014.