

Towards proper selection of primitives and modifications for a cryptographic scheme

Lenka Fibíková

fibikova@exp-math.uni-essen.de

Institute for Experimental Mathematics
University of Essen, Germany

Abstract

Using TST cipher as an example we show how the Random oracle model can be used to discover weak primitives of a cipher without the need for full cryptanalysis, and how to make changes to them in order to get a provable secure scheme.

1 Introduction

It is well known that designing a new cryptographic scheme and giving some arguments to support a belief that the scheme is secure is just a first step in its possible use in practice. Not only in operations but also during implementation phase there is a number of factors that might yield to degradation of the originally satisfying level of security of the scheme. This may be illustrated on cryptographic schemes whose authors focus themselves mainly on the underlying structure of the scheme and basic requirements for the primitives used, while leaving selection of the particular primitives to implementers. However, rarely it is just a structure and some limitations on the primitives that provide good security properties of the whole — more often careful selection from a set of possible primitives is required to obtain secure and useful cryptographic algorithm. Also, sometimes people may be tempted to introduce slight modification to the known scheme, usually for efficiency reasons or in order to further "improve" its security. Thus, even those on the application side of cryptology should think about proper choice of primitives, parameters and possible enhancements and their effect on the security of the underlying cryptographic scheme.

Here we want to point out to one possible approach that allows one to assess how a particular primitive (or modification to the scheme) contributes to the overall security level of the scheme. Particularly we want to show how some ideas from a tool used in theoretical cryptography — namely so called Random oracle model used to argue for security of proposed cryptographic schemes — might be used to discover weak primitives, without the need for full cryptanalysis. Moreover, this approach allows one to compare relative strength of different "good enough" primitives thus enabling to select the best one with respect to security properties.

We illustrate the approach using TST scheme [3] as an example. Namely, we will show that one of hash functions used in the TST scheme is weak and even addition of a permutation to the scheme, proposed in [3], does not provide sufficient compensation for the weakness. Further on, we show that if a good (strong) hash function is used in the scheme, then the permutation does not contribute to the security, and may be removed thus simplifying the scheme. Subsequently we will investigate other hash functions and their use in the scheme and show how one can select the best one with respect to security of the overall scheme.

Rest of the paper is divided as follows. First, rather informal introduction to Random oracle model is given. Here we stress that to show that a primitive is weak it is enough to show existence of respective "distinguisher", i.e. to construct a short algorithm capable to distinguish output of the primitive from an output of a random source. We then give description of the TST scheme as proposed in [3] and its possible simplification. In the next section we describe how to show weakness of a primitive (hash function) used in the scheme and why and how the original TST scheme may be simplified without the loss of its security. Subsequently we also show that our approach allows us to compare relative strength of different hash functions used on the place of the weak one and allows us to pick the best one with respect to security properties (even if all investigated functions are "secure enough").

2 Random Oracle Model

In our security model we will use oracles in two different meanings. The first one is the concept of the random oracle model (ROM). It was formalized by Bellare and Roggaway [1] and its principle is substitution of atomic primitives of a cryptographic scheme by random oracles.

Atomic primitives of a scheme are building blocks, various pseudorandom functions and permutations, which do not solve any specific cryptographic problem, but have to be put together to create the scheme. For example in a Feistel cipher (e.g. DES) the round functions are its atomic primitives. On a higher level, for modes of encryption operation (ECB, CBC, etc.) the whole block cipher may be considered to be an atomic primitive of the mode.

Analyzing security of a scheme together with the specific primitives causes two problems: First, the structure of the primitives brings much complexity into the analysis; second, every change in a primitive demands reevaluation of the whole scheme. The approach of the **random oracle model** is to build a system in two steps. The first step is to design an idealized scheme where all underlying cryptographic primitives are substituted by random oracles (i.e. functions returning truly random outputs), and prove security of this idealized scheme. Next, the random oracles are replaced by "good" cryptographic functions and permutations. In this way, one obtains an implementation of the ideal system in the "real world", where random oracles do not exist.

Since we will use also another type of oracles (see Section 3) in our security proofs of the idealized schemes (schemes in ROM), in the following we will use term "perfect random function" instead of "random oracle".

3 Security Model

While using ROM, we work in rather strict model — the attacker's goal is to distinguish the encryption scheme from a random function in the following game: The attacker has access to an oracle which implements either the encryption scheme or a perfect random function — it is not known to the attacker which one, but it is known that the oracle implements the same function during the whole game. Querying the oracle limited number of times and using its answers and infinite computation power, the attacker has to decide which function the oracle implements, and output 1 ("accept") if it is the encryption scheme, or 0 ("reject") if it is a perfect random function. Since the goal of the attacker is to distinguish two functions, it is called a **distinguisher** [6]. A distinguisher which may query the oracle up to d times is called a **d -limited distinguisher** [6]. In general, the goal of a distinguisher may be to distinguish any two fixed random functions from each other. Different types of distinguishers may be defined, depending on the type of attack they perform. For example:

- a known-plaintext-attack (KPA) distinguisher may query the oracle only with a predefined set of plaintexts and get their ciphertexts;
- a chosen-plaintext-attack (CPA) distinguisher queries the oracle with any plaintexts chosen in advance;
- an adaptive-chosen-plaintext-attack (ACPA) distinguisher may choose any set of plaintexts adaptively depending on all previous answers of the oracle.

Similar distinguishers can be defined for attacks which query the oracle with ciphertexts and get plaintexts (ciphertext-attack distinguishers), as well as attacks which combine choosing plaintexts and ciphertexts (plaintext-ciphertext-attack distinguishers). The adaptive chosen-plaintext-ciphertext-attack (ACPCA) distinguishers are the most powerful ones.

The success of a distinguisher D to distinguish between a cipher F and a random function F^* is determined by two probabilities:

- probability of answering "accept" when the oracle implements F (a correct answer) — p_0 , and
- probability of answering "accept" when the oracle implements F^* (an incorrect answer) — p_1 .

The overall ability of a distinguisher D implementing an attack ATK to distinguish two functions is measured by **advantage** defined as $Adv_D^{\text{ATK}(d)}(F_1, F_2) = |p_0 - p_1|$. It is a value from interval $\langle 0, 1 \rangle$

expressing the probability that the distinguisher is able to distinguish the two functions from each other — a high advantage implies that it can distinguish them with high probability, and vice versa, if the advantage is small, the probability that it distinguishes them is small.

Example 1. Consider a simple distinguisher which always returns "accept". Whenever the oracle implements the function F , the distinguisher "accepts", and thus $p_0 = 1$. Similarly, when a perfect random function is implemented, it also always accepts, and therefore $p_1 = 1$ too. Hence, the advantage of this distinguisher is $|1 - 1| = 0$, which says that the distinguisher cannot distinguish the two functions at all.

The example above shows that an attacker has to use more clever idea than always accepting in order to get nonzero probability of success. However, the goal of a designer of a cryptographic function is to make it difficult for an attacker to distinguish the function from a perfect random one even when the attacker has the very best idea how to attack the function. In other words, the advantage should be very small even for the best possible attack — in the ideal case equal to 0, what means that the attacker cannot do anything better than always accepting.

Note that this model of security is very strong. When one proves that a cipher is secure in this model, then it looks like a truly random function. On the other hand, when one proves that a cipher is not secure in this model, it says only that an attacker has some (high) probability to distinguish that it is not a truly random output. This fact does not say anything about how secure the cipher is against weaker attacks which are more practical for an attacker, like for example finding an encryption key, or being able to encrypt/decrypt another message.

4 Proof of Security

Consider a cryptographic scheme Ω whose implementation involves some cryptographic primitives F_1, \dots, F_r . The function one gets by its implementation will be denoted $F = \Omega[F_1, \dots, F_r]$. The proof of security of the function F can be performed in the following four steps:

1. *Definition of a class of attacks the cipher should be secure against.*

Different types of attacks (distinguishers) can be defined. Besides the general attacks mentioned in the previous section, it is possible to define special subclasses of attacks (differential cryptanalysis, linear cryptanalysis, etc.) in order to get more accurate evaluation of security.

Generally, one has to keep in mind that it may not be enough to consider only one type of attack when dealing with composite ciphers. Particularly, it is necessary to distinguish between attacks against the scheme as a whole and attacks against its primitives (functions). For example, in Feistel ciphers we never need to calculate inverse of the round functions (neither during decryption). Thus, studying a ciphertext attack on a Feistel network, we get a plaintext attack related to the underlying round functions.

In our analysis we will consider two types of attacks — chosen plaintext attacks and adaptive chosen plaintext-ciphertext attacks.

2. *Evaluation of advantage of the scheme in the random oracle model.*

The primitives of the schemes are substituted by perfect random functions, and advantage of $F' = \Omega[F_1^*, \dots, F_r^*]$ is calculated.

3. *Evaluation of advantage of individual cryptographic primitives of the scheme.*

The advantage of individual functions F_1, \dots, F_r is calculated separately. In a more complex scheme, some of them may be decomposed in the similar way as the main scheme; then the advantage may be calculated recursively.

4. *Proof how the advantage of the underlying primitives propagates to the scheme.*

In this step we combine results of Steps 2 and 3 into a final advantage. Assume that calculation of F requires a_i computations of the function F_i ($i = 1, \dots, r$). Then for any d -limited distinguisher

between F and a perfect random function, its advantage is upper-bounded by the sum of advantage of the scheme in ROM (Step 2) and advantages of its primitives ¹ (Step 3) as follows [4]

$$Adv^{\text{ATK}(d)}(F) \leq Adv^{\text{ATK}(d)}(F') + \sum_{i=1}^r Adv^{\text{ATK}_{F_i}(a_i d)}(F_i).$$

Splitting the proof into the steps as described above enables one to separate design of a cipher into several independent tasks and quantify security in term of how much computation of adversary in certain type of attack the cipher can withstand. It also enables one to compare different ciphers as more or less secure than others.

5 TST

TST is a fully scalable cipher based on the Feistel scheme, but unlike the original Feistel scheme, where input is divided into two parts of equal size, input of TST cipher is divided asymmetrically. In principle, there are two possibilities of unbalancing a round of a Feistel cipher. In the first case, the right part (input to the round function) is larger than the left part (output of the round function), and the scheme is called **source heavy** (see Figure 1A), in the other case, the right part is smaller than the left part, and the scheme is called **target heavy** (see Figure 1B) [5]. The TST cipher combines both approaches so that one round of the cipher consist of two unbalanced Feistel rounds (see Figure 1C).

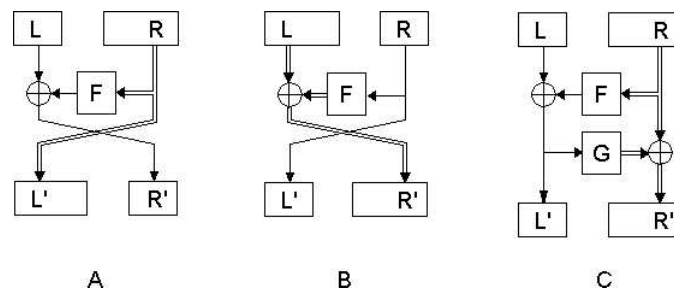
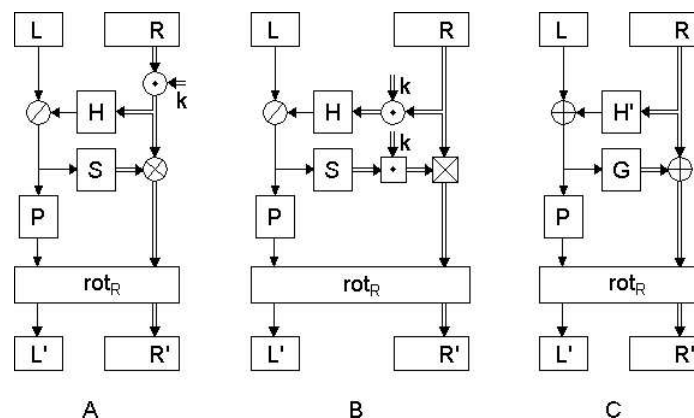


Figure 1: Unbalanced Feistel Network and the TST cipher

Although the TST cipher is based on the mixed unbalanced Feistel scheme, it brings some modifications to it (see Figure 2A): It changes the XOR operation to any invertible operations on binary strings (\odot , \oslash , and \otimes), uses un-keyed round functions with addition of a round key to the right part before each (double-)round, and adds a new permutation to the left part and applies string rotation after each (double-)round. In general, the function H may be any hash function, and S a random sequence generator. [3]



¹As mentioned in Step 1, the attack for the underlying primitives may be different from the attack to the whole scheme — we denote it here as ATK_{F_i} .

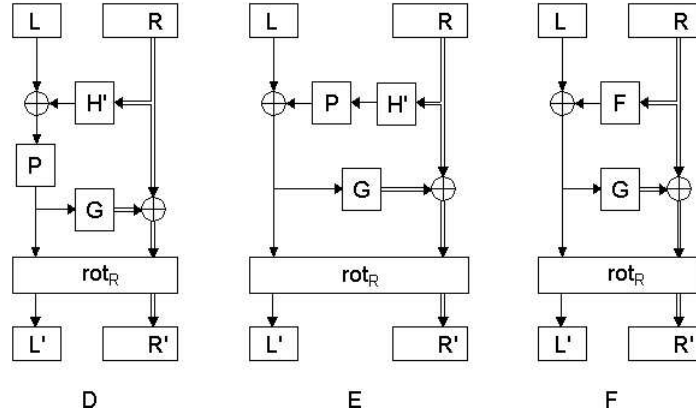


Figure 2: TST and its simplification

Since the function H does not depend on the key (it is not a real primitive of the scheme), we are not able to use the ROM directly. For this reason we make some changes to the scheme which, however, do not weaken its security. First, the addition of the key k (\odot) may be attached to both functions H , and S : $H'(X) = H(X \odot k)$, $G(X) = k \boxtimes S(X)$ (it means that both sub-rounds of a double-round have the same key), and the \otimes is substituted by another operation \boxtimes , where \boxdot and \boxtimes are such that $a \boxtimes (b \boxdot c) = (a \boxdot b) \otimes c$ for all entries (see Figure 2B). Even if such a pair of functions does not exist, one may use $\boxdot = \odot$ and $\boxtimes = \otimes$. This is not equivalent to the original scheme, but we buy flexibility in choice of the functions H' and G which actually introduce randomness into the scheme.

For compatibility with the unbalanced Feistel scheme, we will first consider only XOR function for \odot and \boxtimes . However, in general \odot and \boxtimes may be any randomness-preserving operation (in the sense that if one of the operand is random, the result of the operation remains random) because, as mentioned before, functions H' and G introduce the randomness, and the operations \odot and \boxtimes just transfer the randomness to the data blocks, thus we may do this substitution without loss of generality.

Using these substitutions, one gets the simplified form of TST which actually is a mixed unbalanced Feistel scheme with two additional operations — the permutation P , and the final rotation rot_R (see Figure 2C). Note that the simplification is only in its form. In fact, functions H' and G may be any key-dependent functions with the appropriate input and output sizes, thus the scheme is more general and easier to optimize.

In the following section we shortly discuss security of mixed unbalanced Feistel scheme, ignoring the additions of the TST cipher, and in Section 7 we will argue that these additions are actually redundant and can be removed from the scheme. Here we only shortly sketch the idea why it is so. In the random oracle model, the functions H' , G , and permutation P are substituted by truly random functions and permutation. For any plaintext $[L, R]$, the output $H'(R)$ is then truly random, and thus also $L_1 = L \oplus H'(R)$ is truly random. Similarly, $R_2 = R \oplus G(L_1)$ is truly random. Therefore, in the random oracle model, the permutation P and the final rotation rot_R are redundant. In a real-world implementation of the TST cipher, certainly, the permutation P brings more confusion into the left part, and the final rotation improves diffusion. However, we will show in Section 7 that if the function H' is weak, some plaintext characteristics pass through the permutation P . Moreover, if the contribution of P is significant, it may be clever to put it before the second sub-round (see Figure 2D), in order to bring more randomness to the input of the function G . Further, since \oplus is randomness preserving, P may be put together with H' into one stronger substitution $F = H' \oplus P$ (see Figure 2E, 2F). Therefore, the permutation P is in the general case redundant also in a real-world implementation.

The original scheme of TST (Figure 2A) involves three primitive cryptographic functions: a hash function $H : \{0, 1\}^{nm} \rightarrow \{0, 1\}^m$, a substitution box $S : \{0, 1\}^m \rightarrow \{0, 1\}^{nm}$, and a permutation substitution box $P : \{0, 1\}^m \rightarrow \{0, 1\}^m$. Both S and P are represented by a table of $2^m \times nm$, resp. $2^m \times m$, bits generated by a random bit/number generator (see Figure 3).

Algorithm 1. Generation of S-box S [2]

INPUT: key K
OUTPUT: random S-box S

1. For $i = 0$ to $2^m - 1$ do
 - 1.1 For $j = 0$ to $n - 1$ do
 - 1.1.1 $S_{i,j} = \text{Random}(\{0, 1\}, K)$
 2. Return S
-

Algorithm 2. Generation of S-box P [2]

INPUT: key K
OUTPUT: random S-box P

1. For $i = 0$ to $2^m - 1$ do
 - 1.1 $P_i = i$
 2. For $i = 0$ to $2^m - 1$ do
 - 2.1 $j = \text{Random}(\{i, \dots, 2^m - 1\}, K)$
 - 2.2 $P_i \leftrightarrow P_j$
 3. Return P
-

Figure 3: Generation of Substitution Boxes S and P

In the case the pseudorandom generators used there are unpredictable, it is easy to see that both functions S and P are indistinguishable from perfect random ones, i.e. $Adv^{\text{ATK}(d)}(S) = Adv^{\text{ATK}(d)}(P) = 0$ for any $d > 0$.

In the main part of the paper we concentrate on the choice of the hash function H . Authors of [3] suggest two basic structures depicted here on Figures 4, and 7. For the underlying function T one of the following functions is proposed:

- $f(x) = \text{rot}_c(x)$ where c is relatively prime to m ;
- $f(x) = c \cdot x \bmod 2^m$ where c is an m -bit prime;
- $f(x) = x(2x + 1) \bmod 2^m$;
- or other simple non-linear function.

6 Security of TST Scheme

TST scheme is an unbalanced Feistel network with a few modifications. However, as sketched in the previous section, in ROM it is equivalent to a standard unbalanced Feistel network, and thus they both have equal advantage. For an unbalanced Feistel network Ψ dividing input in m and nm bit parts, for any $d \ll 2^{\frac{m}{2}}$ the following is known [4]:

- 2-round schemes are secure against known plaintext attacks in ROM
($Adv^{\text{KPA}(d)}(\Psi[F_1^*, F_2^*]) \leq \frac{d^2}{2^m}$);
- 2-round schemes are not secure against chosen plaintext attacks;
- 3-round schemes are secure against adaptive chosen plaintext attacks in ROM
($Adv^{\text{ACPA}(d)}(\Psi[F_1^*, F_2^*, F_3^*]) \leq \frac{d^2}{2^m}$);
- 3-round schemes are not secure against adaptive chosen plaintext-ciphertext attacks;
- 4-round UFNs are secure against adaptive chosen plaintext-ciphertext attacks in ROM
($Adv^{\text{ACPCA}(d)}(\Psi[F_1^*, F_2^*, F_3^*, F_4^*]) \leq \frac{d^2}{2^m}$).

For security against an attacker with access to more plaintext/ciphertext pairs, the number of rounds has to be increased to at least $a \frac{l-1}{m-1-2 \lg d}$, where 2^{-l} is the requested upper-bound for advantage, and $a = 2$ for known plaintext attack, 3 for chosen ciphertext attacks, or 4 for adaptive chosen plaintext-ciphertext attacks. Concerning the value of l , [4] argues that reasonable value of l is $\min\{\text{keysize}, \text{blocksize}\}$ because then the probability of successful distinguishing is roughly the same as of the pure guessing.

7 Analysis of TST Primitives

As mentioned in Section 5 when the pseudorandom generators are ideal both functions S and P are indistinguishable from perfect random ones. Therefore, we will further analyze only different schemes for the hash function H .

First, consider the hierarchical structure H_1 — one of the schemes proposed in [3] — depicted on Figure 4.

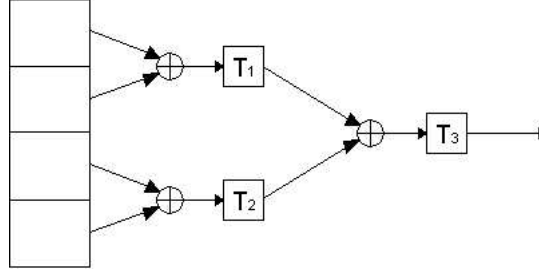


Figure 4: Hierarchical Structure H_1

Since the functions T_i are executed only after the XOR functions, any two inputs containing two blocks leading into the same execution of T_i with equal XOR cause equal outputs of the hash function. Thus we can build the following distinguisher.

Distinguisher 1. (D_1): 2-limited CPA distinguisher for H_1

1. Create $X_1 = (x_1, x_2, x_3, \dots, x_n)$ at random.
2. Create $X_2 = (x_1 \oplus c, x_2 \oplus c, x_3, \dots, x_n)$ for a non-zero constant c .
3. Query the oracle with X_1 and X_2 , and get Y_1 and Y_2 , where Y_i is either $H_1(X_i)$ or $F^*(X_i)$.

If $Y_i = H_1(X_i)$ then XORing the first two blocks in both of the plaintext messages eliminates the constant c ($x_1 \oplus x_2 = x_1 \oplus c \oplus x_2 \oplus c$) and all T_i s are executed with the same arguments for both X_1 and X_2 . Therefore $Y_1 = Y_2$.

4. If $Y_1 = Y_2$ then output "accept".
 5. Output "reject".
-

When the oracle implements H_1 , the distinguisher D_1 always answers correctly, i.e. $p_0 = 1$. When the oracle implements a perfect random function, probability that two random strings of length m are equal is $p_1 = \frac{1}{2^m}$. Therefore advantage of this 2-limited distinguisher is

$$Adv_{D_1}^{\text{CPA}(2)}(H_1) = |p_0 - p_1| = 1 - \frac{1}{2^m}.$$

The advantage may be increased by adding further chosen plaintexts in similar way as X_2 , so that $Adv^{\text{CPA}(d)}(H_1) \geq 1 - \frac{1}{2^{m(d-1)}}$. Thus, the hierarchical structure H_1 is not secure against chosen plaintext attacks. In the following we show in more details how this attack can be extended to the whole TST scheme and that the additional S-box P cannot stop the attack.

Consider the difference between two inputs of a TST round (consisting of one m -bit block for the left part and n m -bit blocks for the right part) X_1 and X_2 . If the attacker creates two messages $X_1 = (x_0, x_1, x_2, x_3, \dots, x_n)$, and $X_2 = (x_0, x_1 \oplus c, x_2 \oplus c, x_3, \dots, x_n)$, their differential characteristic is $(0, c, c, 0, \dots, 0)$, the right parts (without x_0) are led into the function H_1 . Let h_i be the output of the function H_1 for X_i . As shown in the previous distinguisher, $h_1 = h_2$, and thus also $x_0 \oplus h_i$, $P(x_0 \oplus h_i)$, as well as $S(x_0 \oplus h_i)$ give the same values for both inputs. Therefore, the outputs of one TST round have the same difference as the inputs, i.e. $(0, c, c, 0, \dots, 0)$ — see also Figure 5. If there were no rotation in the scheme, the characteristic would spread through the whole cipher.

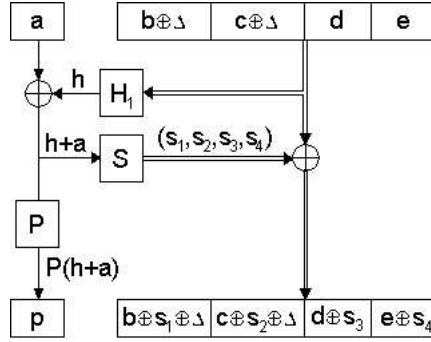


Figure 5: Characteristic Propagation

In the original design of TST a round key is added to the right part of plaintext, and only then it is hashed. If the key is XORed with the plaintext, the changes in the second message are eliminated as described above. Author of [2] suggested to use modular addition operation to combine round keys with messages in order to make an attack more difficult. However, we will show that this improvement is not sufficient against the distinguishing attacks. Consider the following distinguisher between H_1 and a perfect cipher used on a plaintext with added key:

Distinguisher 2. (D_2): 2-limited CPA distinguisher for H_1

1. Create $X_1 = ((0 \dots 00)_2, (0 \dots 00)_2, x_3, \dots, x_n)$ at random.
 2. Create $X_2 = ((0 \dots 01)_2, (0 \dots 01)_2, x_3, \dots, x_n)$.
 3. Query the oracle with X_1 and X_2 and get Y_1 and Y_2 , where Y_i is either $H_1(X_i)$ or $F^*(X_i)$.
 If $Y_i = H_1(X_i)$, and the number of ones in the least significant bits followed by 0 are equal in both the first and second block of the key (0-condition) — this occurs with probability at least $1/4$ (what is probability that both least significant bits are 0) — the XOR operation eliminates the ones in the least significant bits, and all T_i s are executed with the same arguments for both X_1 and X_2 . Therefore, $Y_1 = Y_2$.
 4. If $Y_1 = Y_2$ then output "accept".
 5. Output "reject".
-

If the round key is chosen randomly, probability that the 0-condition holds is $q = \frac{1}{3} + 3\left(\frac{1}{4}\right)^{m+1} \geq \frac{1}{4}$. In this case if the oracle implements H_1 , the distinguisher D_2 always answers correctly, i.e. $p_0 = 1$, and advantage of this distinguisher is

$$Adv_{D_2}^{CPA(2)}(H_1) = |p_0 - p_1| \geq 1 - \frac{1}{2^m}.$$

If the oracle implements H_1 , the distinguisher D_2 is successful only when the key satisfied the 0-condition, and since the key is not changed during the attack, there is quite high probability $(1 - q)$ that it will not work. However, when the attack fails (the distinguisher D_2 rejects), the attack may be repeated similarly by setting 1 for another bit of the first and second block of X_2 . Even when the distinguisher gives the rejecting answer for all bits of the pairs from the first and second block (0-condition failed), the attacker can continue in the same way working with the third and fourth block of X_1 , etc. If the key is chosen randomly, probability that the key does not satisfy the 0-condition on all its bits is less then $\left(\frac{3}{4}\right)^{n/2}$.

After the distinguisher first time accepts it is convenient to make several rounds with the same choice of bits for the 0-condition in order to eliminate possibility of accepting a perfect random function. However, it may also happen that the distinguisher first accepts although the key does not satisfy the 0-condition (it occurs when T_1 returns the same value on different inputs) and later rejects, thus wrongly identifies the function as a truly random one. Probability that the distinguisher accepts l times although the key does not satisfy the 0-condition is $\frac{1}{2^{ml}}$ and therefore $p_0 = 1 - \frac{1}{2^{m(d-k+1)}}$.

If the oracle implements a perfect random function, it may be wrongly identified as H_1 when after the first accepting answer, the distinguisher only accepts, i.e. $p_1 = \frac{1}{2^{m(d-k+1)}}$. Therefore,

$$Adv_D^{CPA(d)}(H_1) \geq 1 - \frac{1}{2^{m(d-k+1)-1}}.$$

It means that the introduction of the addition operation, as suggested in [3], brings improvement on using only XOR function, but the advantage still remains high.

Now we take a look at the second structure of the hash function suggested in [2]. It is the scheme depicted on Figure 7, using a weak function f for T . This is actually the hash function which was implemented and tested by the author of [2]. There we can apply the following attack:

Distinguisher 3. (D_3): 2-limited CPA distinguisher for H_2

1. Create $X_1 = (x_1, x_2, x_3, \dots, x_n)$ at random.
2. Create $X_2 = (x_1 \oplus a, x_2 \oplus c, x_3, \dots, x_n)$ for any two constants a and c such that $f(x_1 \oplus a) = f(x_1) \oplus c$ and at least one of them is nonzero.
3. Query the oracle with X_1 and X_2 and get Y_1 and Y_2 , where Y_i is either $H_2(X_i)$ or $F^*(X_i)$.

If $Y_i = H_2(X_i)$ then the calculation for the message X_1 follows this calculation sequence:

$$\begin{aligned} r_1 &= f(x_1) \\ r_i &= f(r_{i-1} \oplus x_i) \quad \text{for all } i = 2, \dots, n \end{aligned}$$

The second calculation sequence starts with:

$$\begin{aligned} r'_1 &= f(x_1 \oplus a) = f(x_1) \oplus c = r_1 \oplus c \\ r'_2 &= f(r_1 \oplus c \oplus x_2 \oplus c) = f(r_1 \oplus x_2) = r_2 \end{aligned}$$

and thus in all further steps they both follow the same computation sequence. Therefore, $Y_1 = Y_2$.

4. If $Y_1 = Y_2$ then output "accept".
 5. Output "reject".
-

When the oracle implements H_2 , the distinguisher D_3 always answers correctly, i.e. $p_0 = 1$, and therefore advantage of this 2-limited distinguisher is again

$$Adv_{D_3}^{\text{CPA}(2)}(H_2) = |p_0 - p_1| = 1 - \frac{1}{2^m}.$$

The advantage may be increased by adding further chosen plaintexts in similar way as X_2 , so that $Adv^{\text{CPA}(d)}(H_2) \geq 1 - \frac{1}{2^{m(d-1)}}$.

In order to increase security of the hash function, author of [2] suggested a double-scheme consisting of two executions of the hash function H_2 — once as described above, and once with a shifted message — and finally XORing their outputs, i.e. $H'_2(x_1, \dots, x_n) = H_2(x_1, \dots, x_n) \oplus H_2(x_n, x_1, \dots, x_{n-1})$ (see Figure 6). However, when the messages are constructed as in the distinguisher D_3 , the difference is eliminated when the modified blocks are on any place in the message provided they follow each other as described. Thus, the double structure does not bring any improvement against this distinguishing attack.

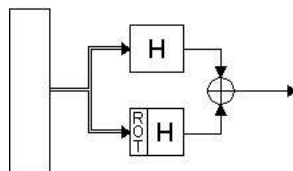


Figure 6: Double Hash Function

Since we are able to create two distinct messages with the same output of H_2 , the remark in the previous section about spreading the characteristic through the round holds also for this hash scheme. A distinguisher for the hash scheme with modular addition of a round key to the message can be created as well.

The results described above do not imply that there is no appropriate hash scheme for TST. In what follows we show how careful analysis of various candidates and subsequent modifications allows us to find schemes with small advantage and choose the best one.

Consider the same structure as H_2 , but with perfect random function T^* as the primitive (let denote it H_3). In the following we show that advantage of any attacker trying to distinguish between output of this scheme and a perfect random one is small [7].

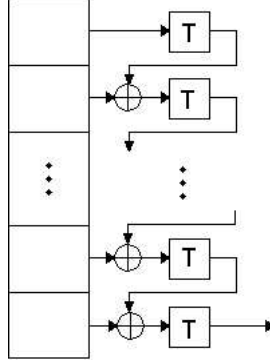


Figure 7: Serial Structure H_2-H_4

We may assume that inputs for the function the attacker asks oracle are always different (pairwise different) because asking the oracle twice with the same input does not bring any new information. If the oracle implements a perfect random hash function, all outputs are equiprobable. If the oracle implements the function H_3 and inputs to the last instance of T^* are all different then there is always a perfect random function which returns the same outputs as the function H_3 , and thus they are indistinguishable. Therefore, even the best distinguisher which is able to distinguish the function H_3 from the perfect random one in all other cases cannot have advantage better than $1 - \Pr[\text{all inputs to the last instance of } T^* \text{ are different}]$. Let $I_{i,j}$ is input to the j -th instance of T^* for the i -th plaintext. Then

$$Adv^{\text{ACPA}(d)}(H_3) \leq 1 - \Pr[\forall 1 \leq i < j \leq d : I_{i,n} \neq I_{j,n}] = \Pr[\exists i \neq j : I_{i,n} = I_{j,n}].$$

Let inputs to the hash function are $X_i = (x_{i,1}, \dots, x_{i,n})$, and corresponding outputs are $Y_i = (y_{i,1}, \dots, y_{i,n})$ (for $1 \leq i \leq d$). Let call the inputs collision free when during all dn computations of T^* all outputs of T^* are pairwise different whenever the inputs to the functions are different. If the inputs of the hash function are collision free, then probability that two inputs to the last instance of T^* ($I_{i,n} = I_{j,n}$) are equal is

$$\begin{aligned} \Pr[I_{i,n} = I_{j,n}] &= \Pr[y_{i,n-1} \oplus x_{i,n} = y_{j,n-1} \oplus x_{j,n}] \\ &= \begin{cases} \Pr[y_{i,n-1} = y_{j,n-1}] \stackrel{\text{ColFree}}{=} \Pr[I_{i,n-1} = I_{j,n-1}] & \text{if } x_{i,n} = x_{j,n} \\ \Pr[y_{i,n-1} = y_{j,n-1} \oplus x_{i,n} \oplus x_{j,n}] & \text{if } x_{i,n} \neq x_{j,n} \end{cases} \end{aligned}$$

Thus if r is the smallest integer such that $x_{i,n-r} \neq x_{j,n-r}$ (note that $r \geq 1$, because $X_i \neq X_j$) then

$$\begin{aligned} \Pr[I_{i,n} = I_{j,n}] &= \Pr[I_{i,n-1} = I_{j,n-1}] = \dots = \Pr[I_{i,n-r} = I_{j,n-r}] \\ &= \Pr[y_{i,n-r-1} = y_{j,n-r-1} \oplus x_{i,n-r} \oplus x_{j,n-r}] \\ &= \begin{cases} 0 & \text{if } I_{i,n-r-1} = I_{j,n-r-1} \\ \frac{1}{2^m} & \text{if } I_{i,n-r-1} \neq I_{j,n-r-1} \end{cases} \\ &\leq \frac{1}{2^m}. \end{aligned}$$

Combining the previous results we get

$$Adv^{ACPA(d)}(H_3) \leq \frac{(dn)^2}{2^{m+1}}.$$

The non-collision condition above has to be rather strong, and causes quite high advantage. This can be improved by using different random functions T_1^*, \dots, T_n^* for each m -bit block of the input (let denote the scheme as H_4). In this case the upper-bound of the advantage decreases to

$$Adv^{ACPA(d)}(H_4) \leq \frac{dn^2}{2^{m+1}}.$$

We have already shown that message blocks in the hierarchical structure H_1 should **not** be XORed before execution of functions T_i . The simplest way how to avoid this is to apply the functions on the message blocks first, and then simply XOR the results together (see Figure 8), thus getting the scheme H_5 .

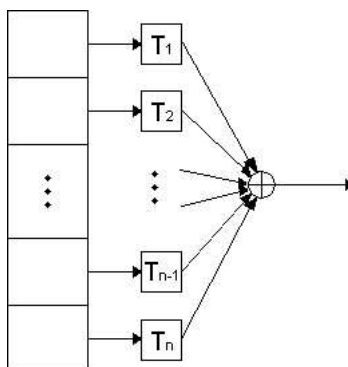


Figure 8: Parallel Structure H_5

This scheme is indistinguishable from a perfect random one as long as an attacker cannot obtain more than three plaintext/ciphertext pairs, i.e. $Adv^{ATK(3)}(H_5) = 0$. However, when the attacker may choose plaintext messages, it is possible to distinguish this scheme using the following distinguisher:

Distinguisher 4. (D_4): 4-limited CPA distinguisher for H_5

1. Create $X_1 = (s, u, x_3, \dots, x_n)$ at random.
 2. Create $X_2 = (s, v, x_3, \dots, x_n)$ so that $u \neq v$.
 3. Create $X_3 = (t, u, x_3, \dots, x_n)$ so that $s \neq t$.
 4. Create $X_4 = (t, v, x_3, \dots, x_n)$.
 5. Query the oracle with X_1, X_2, X_3 and X_4 , and get Y_1, Y_2, Y_3 and Y_4 , where Y_i is either $H_5(X_i)$ or $F^*(X_i)$.
- If $Y_i = H_5(X_i)$ then

$$\begin{aligned} T_1^*(s) \oplus T_2^*(u) \oplus T_3^*(x_3) \dots \oplus T_n^*(x_n) &= Y_1 \\ T_1^*(s) \oplus T_2^*(v) \oplus T_3^*(x_3) \dots \oplus T_n^*(x_n) &= Y_2 \\ T_1^*(t) \oplus T_2^*(u) \oplus T_3^*(x_3) \dots \oplus T_n^*(x_n) &= Y_3 \\ T_1^*(t) \oplus T_2^*(v) \oplus T_3^*(x_3) \dots \oplus T_n^*(x_n) &= Y_4 \end{aligned}$$

Therefore, $0 = Y_1 \oplus Y_2 \oplus Y_3 \oplus Y_4$.

6. If $Y_1 \oplus Y_2 \oplus Y_3 \oplus Y_4 = 0$ then output "accept".
 7. Output "reject".
-

When the oracle implements H_5 , the distinguisher D_4 always answers correctly, i.e. $p_0 = 1$. When the oracle implements a perfect random function, probability that XOR of four random strings of length m gives 0 is $p_1 = \frac{1}{2^m}$. Therefore advantage of this 4-limited distinguisher is

$$Adv_{D_4}^{CPA(4)}(H_5) = |p_0 - p_1| = 1 - \frac{1}{2^m}.$$

The advantage may be increased by adding further chosen plaintexts in similar way as X_3 and X_4 , so that $Adv^{CPA(2d)}(H_5) \geq 1 - \frac{1}{2^{m(d-1)}}$.

Thus this scheme is not secure when the attacker can obtain more than three plaintext/ciphertext pairs. However, by adding another random function to the output of the scheme (see Figure 9), we can achieve small advantage also for bigger d ($Adv^{ATK(d)}(H_6) \leq \frac{d^2}{2^{m+1}}$) while retaining the zero advantage for $d \leq 3$.

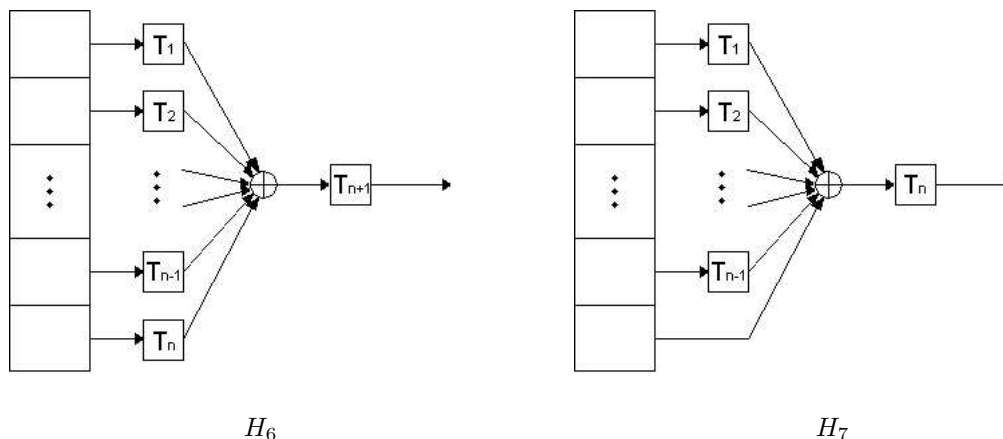


Figure 9: Parallel Structures H_6 and H_7

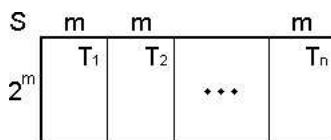
This scheme has a disadvantage comparing to all previous hash schemes — it is less efficient due to the additional random function. However, one of the first-level functions (see Figure 9) may be omitted without increasing the upper bound of its advantage (i.e. $Adv^{ATK(d)}(H_7) = Adv^{ATK(d)}(H_6)$ for all d).

8 Conclusions

In the TST cipher as proposed in [3] both S-boxes S and P are perfect random if they are generated by a perfect random bit/number generator. However, the perfect pseudorandomness of the S-box P does not help when the hash scheme is weak, or when it is built on a weak function. On the other hand, when the scheme and the function the scheme is built on are both strong, the output of the hash function is close to perfect random, and the S-box P is not necessary.

The hash function H_2 recommended in [2] is weak, although we have not shown how its weakness can be exploited (even if we were able to find a distinguisher which distinguish output of TST from a perfect random one, it would not tell us how to construct an attack which reconstructs a key, or at least which encrypts or decrypts another message). However, by a sequence of carefully analyzed improvements we have found another hash scheme H_7 which is provable secure — it is perfect random when the attacker can obtain up to 3 plaintext/ciphertext pairs, and for attacks of larger sizes the probability of distinguishing it from a perfect random function is still small if the size of the attack $d \ll 2^{\frac{nm}{2}}$.

The random functions T_1, \dots, T_n for H_7 can be generated using a random bit generator in the same way as the S-box S , and stored in tables of size $2^m \times m$. Since the S-box S is represented by $2^m \times nm$ table, if memory space is important the table of S can be divided into n parts, so that each function T_i is defined by $(i-1)m$ -th ... $(im-1)$ -th columns of the table, as depicted on the following figure.



When weak primitives were used, the rotation at the end of each round stopped propagation of the characteristic, and application of modular addition for adding key decreased the advantage of the hash scheme, thus they can improve security of the whole scheme when the generator is not perfect. Therefore it may be advantageous to leave them in the scheme although they are redundant in ROM.

Acknowledgments

The author is grateful to Jozef Vyskoč for many helpful comments.

References

- [1] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.
- [2] Valer Canda. *Scalable Symmetric Ciphers Based on Group Bases*. PhD thesis, Institute of Experimental Mathematics, University of Essen, 2001.
- [3] Valer Canda and Trung van Tran. Scalable block ciphers based on Feistel-like structure. Technical report, Institute of Experimental Mathematics, <http://www.exp-math.uni-essen.de/preprints/2001/Iterative.ps>, 2000.
- [4] Lenka Fibikova. *Provably Secure Scalable Block Ciphers, in preparation*. PhD thesis, Institute of Experimental Mathematics, University of Essen.
- [5] Bruce Schneier and John Kelsey. Unbalanced Feistel networks and block-cipher design. *Lecture Notes in Computer Science*, 1039:121–144, 1996.
- [6] Serge Vaudenay. Provable security for block ciphers by decorrelation. In *Symposium on Theoretical Aspects of Computer Science*, pages 249–275, 1998.
- [7] Serge Vaudenay. On provable security for conventional cryptography. In *Information Security and Cryptology ICISC'99*, pages 1–16. Springer-Verlag, 1999.

Appendix

Theorem 1. Let H_4 be a function from $\{0, 1\}^{nm}$ to $\{0, 1\}^m$ defined for any $X = (x_1, \dots, x_n) \in \{0, 1\}^{nm}$ as

$$H_4(x_1, \dots, x_n) = T_n^*(T_{n-1}^*(\dots T_1^*(x_1) \oplus x_2 \dots) \oplus x_n),$$

where T_i^* s are independent perfect random functions from $\{0, 1\}^m$ to $\{0, 1\}^m$ (see Figure 7). Then for any integer d ,

$$Adv^{ACPA(d)}(H_4) \leq \frac{nd^2}{2^{m+1}}.$$

Proof: The proof is similar to the proof for H_3 , with difference that ColFree is defined as an event that for all pairs of inputs in the same column $I_{i,a}$ and $I_{j,a}$ such that $1 \leq i, j \leq d$, and $1 \leq a < n$, if $I_{i,a} \neq I_{j,a}$ then $y_{i,a} \neq y_{j,a}$. (Since T_i^* are independent there is no intercolumnar dependency). Thus,

$$\Pr[\overline{\text{ColFree}}] \leq (n-1) \binom{d}{2} \frac{1}{2^m}$$

and

$$\begin{aligned} Adv^{ACPA(d)}(H_4) &\leq \Pr[\exists i, j : I_{i,n} = I_{j,n} \wedge \text{ColFree}] + \Pr[\overline{\text{ColFree}}] \\ &\leq \binom{d}{2} \frac{1}{2^m} + \binom{d}{2} \frac{n-1}{2^m} \\ &\leq \frac{nd^2}{2^{m+1}} \end{aligned}$$

■

Theorem 2. Let H_5 be a function from $\{0, 1\}^{nm}$ to $\{0, 1\}^m$ defined for any $X = (x_1, \dots, x_n) \in \{0, 1\}^{nm}$ as

$$H_5(x_1, \dots, x_n) = \bigoplus_{a=1}^n T_a^*(x_a),$$

where T_i^* are independent perfect random functions from $\{0, 1\}^{nm}$ to $\{0, 1\}^m$ (see Figure 8). Then for all $d \leq 3$,

$$Adv^{\text{ATK}(d)}(H_5) = 0.$$

Proof: Without loss of generality assume that X_1, X_2, \dots, X_d are all different. Let X_i consists of n blocks, i.e.

$$X_i = x_{i,1}, \dots, x_{i,n}. \text{ Let } z_{i,k} = T_k^*(x_{i,k}).$$

If $d = 1$:

$$\begin{aligned} \Pr[H_5(x_1) = y_1] &= \sum_{\substack{z_{1,1}, \dots, z_{1,n} \\ \bigoplus_{a=1}^n z_{1,a} = y_1}} \Pr \left[\bigwedge_{a=1}^n T_a^*(x_{1,a}) = z_{1,a} \right] \\ &= \sum_{\substack{z_{1,1}, \dots, z_{1,n} \\ \bigoplus_{a=1}^n z_{1,a} = y_1}} \prod_{a=1}^n \Pr[T_a^*(x_{1,a}) = z_{1,a}] \\ &= 2^{m(n-1)} \cdot \frac{1}{2^{mn}} = \frac{1}{2^m} = \Pr[F^*(x_1) = y_1] \end{aligned}$$

If $d = 2, 3$:

Assume that for a -th block, there are $k_a \in \{0, \dots, d\}$ different values among $x_{1,a}, x_{2,a}, \dots, x_{d,a}$. Since all T_a s are independent, for any $z_{1,a}, z_{2,a}, \dots, z_{d,a}$

$$\Pr[\forall i \leq d : T_a^*(x_{i,a}) = z_{i,a}] = \begin{cases} 0 & \exists i, j : x_{i,a} = x_{j,a} \wedge z_{i,a} \neq z_{j,a} \\ \frac{1}{2^{mk_a}} & \text{otherwise} \end{cases}$$

Consider first the case that $d = 3$. We can choose random $z_{i,a}$ without having a collision (thus having the non-zero probability) in the following way: There must be such i and a that for $x_{i,b} \neq x_{j,b}$ both $j \in \{1, 2, 3\} \setminus \{i\}$, else all three inputs were equal. Without loss of generality we may assume that $i = 3$. For all $z_{1,1}, \dots, z_{1,d-1}$ we may set any value from \mathcal{M} . The last one is calculated from the previous ones and $\bigoplus_{a=1}^n z_{1,a} = y_1$. All equations $x_{1,a} = x_{2,a}$ induce $z_{2,a} := z_{1,a}$. However, there must be at least one free $x_{2,a}$, which can be calculated from $\bigoplus_{a=1}^n z_{2,a} = y_2$ after setting other free positions at random. For $i = 3$, first all equations with $j = 1, 2$ are set, all free positions except of $x_{3,b}$ are chosen at random, and $x_{3,b}$ is calculated from $\bigoplus_{a=1}^n z_{3,a} = y_3$. Since in each block, there are k_a different values among $x_{1,a}, x_{2,a}, \dots, x_{d,a}$, there are altogether $\sum_{a=1}^n (3 - k_a) + d$ fixed values. In the similar way we get that if $d = 2$, there are $\sum_{a=1}^n (2 - k_a) + d$ values. Therefore,

$$\begin{aligned} \Pr \left[\bigwedge_{i=1}^d H_4^*(x_i) = y_i \right] &= \sum_{\substack{i=2,3; z_{i,1}, \dots, z_{i,n} \\ \bigoplus_{a=1}^n z_{i,a} = y_i}} \Pr \left[\bigwedge_{i=1}^d \bigwedge_{a=1}^n T_a^*(x_{i,a}) = z_{i,a} \right] \\ &= \sum_{\substack{i=2,3; z_{i,1}, \dots, z_{i,n} \\ \bigoplus_{a=1}^n z_{i,a} = y_i}} \prod_{a=1}^n \Pr \left[\bigwedge_{i=1}^d T_a^*(x_{i,a}) = z_{i,a} \right] \\ &= \sum_{\substack{i=2,3; z_{i,1}, \dots, z_{i,n} \\ \bigoplus_{a=1}^n z_{i,a} = y_i \\ a(d-k_a \text{ equations}): z_{i,a} = z_{j,a}}} \prod_{a=1}^n \frac{1}{2^{mk_a}} \\ &= 2^{m((n-1) \cdot d - \sum_{a=1}^n (d-k_a))} \cdot 2^{-m \sum_{a=1}^n k_a} = \frac{1}{2^{md}} \end{aligned}$$

Therefore, H_5 has the perfect 3-wise decorrelation, and it is not possible to distinguish it from a truly random function seing up to three plaintext/ciphertext pairs. ■

Theorem 3. Let H_6 be a function from $\{0, 1\}^{nm}$ to $\{0, 1\}^m$ defined for any $X = (x_1, \dots, x_n) \in \{0, 1\}^{nm}$ as

$$H_6(x_1, \dots, x_n) = T_{n+1}^* \left(\bigoplus_{a=1}^n T_a^*(x_a) \right),$$

where T_1^*, \dots, T_n^* are independent perfect random functions from $\{0, 1\}^m$ to $\{0, 1\}^{m'}$, and T_{n+1}^* is a perfect random function from $\{0, 1\}^{m'}$ to $\{0, 1\}^m$. Then for any integer d ,

$$Adv^{\text{ATK}(d)}(H_6) \leq \frac{d^2}{2^{m+1}}.$$

Proof: If all inputs to T_{n+1}^* are pairwise distinct, then the outputs Y_i s are perfect random, since the function T_{n+1}^* is perfect random. Let Z_i denote input into T_{n+1}^* for i -th message. Then

$$\Pr[Z_i = Z_j] = \Pr \left[\bigoplus_{a=1}^n T_a^*(X_{i,a}) = \bigoplus_{a=1}^n T_a^*(X_{j,a}) \right] = \frac{1}{2^m}$$

Hence,

$$\begin{aligned} Adv^{\text{ACPA}(d)}(H_6) &= 1 - \Pr[\forall i, j : Z_i \neq Z_j] = \Pr[\exists i, j : Z_i = Z_j] \\ &\leq \sum_{1 \leq i < j \leq d} \Pr[Z_i = Z_j] = \binom{d}{2} \frac{1}{2^m} \end{aligned}$$
■

Theorem 4. Let H_7 be a function from $\{0, 1\}^{nm}$ to $\{0, 1\}^m$ defined for any $X = (x_1, \dots, x_n) \in \{0, 1\}^{nm}$ as

$$H_7(x_1, \dots, x_n) = T_n^* \left(x_n \oplus \bigoplus_{a=1}^{n-1} T_a^*(x_a) \right),$$

where T_1^*, \dots, T_n^* are independent perfect random functions from $\{0, 1\}^m$ to $\{0, 1\}^m$. Then for any integer d ,

$$Adv^{\text{ATK}(d)}(H_7) \leq \frac{d^2}{2^{m+1}}.$$

Proof: If all inputs to T_n^* are pairwise distinct, then the outputs are perfect random, since the function T_n^* is perfect random.

Assume that all inputs are pairwise distinct. Consider two cases: First, when in a pair of plaintexts X_i, X_j there are all blocks up to the last one equal, i.e. when for all $1 \leq a < n$, $x_{i,a} = x_{j,a}$. Then $x_{i,n} \neq x_{j,n}$. Let

$$A := \bigoplus_{a=1}^{n-1} T_a^*(x_{i,a}) = \bigoplus_{a=1}^{n-1} T_a^*(x_{j,a}).$$

$$\Pr[Z_i = Z_j] = \Pr[x_{i,n} \oplus A = x_{j,n} \oplus A] = 0$$

If there is $r < n$ such that $x_{i,r} \neq x_{j,r}$ then

$$\begin{aligned} \Pr[Z_i = Z_j] &= \Pr \left[T_r^*(x_{i,r}) = T_r^*(x_{j,r}) \oplus x_{i,n} \oplus x_{j,n} \oplus \bigoplus_{\substack{1 \leq a < n \\ a \neq r}}^n T_a^*(x_{i,a}) \oplus \bigoplus_{\substack{1 \leq a < n \\ a \neq r}} T_a^*(x_{j,a}) \right] \\ &= \frac{1}{2^m} \end{aligned}$$

Hence,

$$\begin{aligned} Adv^{\text{ATK}(d)}(H_6) &= 1 - \Pr[\forall i, j : Z_i \neq Z_j] = \Pr[\exists i, j : Z_i = Z_j] \\ &\leq \sum_{1 \leq i < j \leq d} \Pr[Z_i = Z_j] \leq \binom{d}{2} \frac{1}{2^m} \end{aligned}$$

■