A Practical Key Recovery Attack on Basic TCHo [*] [**]

Mathias Herrmann[1] and Gregor Leander[2]

[1] Horst Görtz Institute for IT-Security
Faculty of Mathematics
Ruhr-University Bochum
Germany
mathias.herrmann@rub.de

[2] Department of Mathematics
Technical University of Denmark
Denmark
g.leander@dtu.mat.dk

Abstract. TCHo is a public key encryption scheme based on a stream cipher component, which is particular suitable for low cost devices like RFIDs. In its basic version, TCHo offers no IND-CCA2 security, but the authors suggest to use a generic hybrid construction to achieve this security level. The implementation of this method however, significantly increases the hardware complexity of TCHo and thus annihilates the advantage of being suitable for low cost devices. In this paper we show, that TCHo cannot be used without this construction. We present a chosen ciphertext attack on basic TCHo that recovers the secret key after approximately $d^{3/2}$ decryptions, where d is the number of bits of the secret key polynomial. The entropy of the secret key is $\log_2 \binom{d}{w}$, where w is the weight of the secret key polynomial, and w is usually small compared to d. In particular, we can break all of the parameters proposed for TCHo within hours on a standard PC.

Keywords. TCHo , chosen ciphertext attack, stream cipher

1 Introduction

Since the invention of public key cryptography many different crypto systems have been presented. The most popular systems are either based on the hardness of factoring large integers or related problems (e.g. RSA) or computing discrete logarithms in various groups (e.g. DSA, ECDSA). While these schemes are an excellent and preferred choice in almost all applications, there is still a strong need for alternative systems based on other (supposedly) hard problems. This is mainly due to the following two reasons. The first reason is that most

[*] The work described in this paper has been supported by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II.

[**] This research was partly supported by the German Research Foundation (DFG) as part of the project MA 2536/3-1.

of the standard schemes are not suitable for very constraint environments like RFID tags and sensor networks. This problem becomes even more pressing when looking at the next century's IT landscape, where a massive deployment of tiny computer devices is anticipated and thus the need for extremely low cost public key cryptography will increase significantly. The second — and quite unrelated — reason is that most popular public key crypto systems like RSA, DSA and ECDSA will be broken if quantum computers with sufficiently many qubits can be built (see [9]). Thus, it is important to search for public key crypto systems that have the potential to resist future quantum computers.

Those two reasons outlined above inspired Finiasz and Vaudenay to develop the crypto system TCHo [7]. The original version of TCHo has been revised by Aumasson, Finiasz, Meier and Vaudenay (see [3]). This revision was done mainly to improve the efficiency of the original scheme and we refer to TCHo as defined in [3].

TCHo is a public key encryption scheme based on a stream cipher component. TCHo uses mainly hardware friendly operations and is therefore suitable for low cost devices. Its security is based on the problem of finding a low weight multiple of a given polynomial in $\mathbb{F}_2[x]$ (LWPM for short). The public key of TCHo is a high degree polynomial $P \in \mathbb{F}_2[x]$ and the secret key K is a sparse, or low weight, multiple of P. The LWPM problem is of importance in syndrome decoding [5], stream cipher analysis and efficient finite field arithmetics [4]. In [2] El Aimani and von zur Gathen provide an algorithm to solve this problem based on lattice basis reduction and furthermore give an overview of other possible approaches to tackle LWPM. Yet, the suggested parameters of TCHo cannot be broken by any of those attacks.

In this paper we present a chosen ciphertext attack on TCHo that recovers the secret key after roughly $d^{3/2}$ decryptions, where d is the degree of the secret polynomial. In particular all proposed parameters of TCHo given in [3] can be broken within hours on a standard PC. Our attack recovers consecutively all bits of the secret key by decrypting pairs of ciphertexts with a carefully chosen difference. The choice of the difference as well as the choice of the ciphertext depend on all key bits recovered so far. This property of our attack is of independent interest, as it is one of the rare occasions where an attack on a public key crypto system is actually inherently adaptive with respect to the information gained so far. To clarify, we do not solve the problem of finding low weight multiples of a given polynomial efficiently, but rather provide an efficient method to extract this low weight polynomial given a decryption oracle.

It should be noted that the designers do not claim that TCHo is IND-CCA2 secure. On the contrary, as shown in [3] TCHo is clearly not IND-CCA2 secure since it is, just like RSA, trivially malleable. Given an encryption y for a message m and a second message m', it is easy to construct an encryption for $m \oplus m'$. However, as opposed to the trivial IND-CCA2 attack on TCHo that recovers the message our CCA1 attack recovers the secret key.

In [3] the authors propose to use the revised Fujisaki-Okamoto [8] construction from [1] to transform TCHo into a IND-CCA secure scheme. Clearly, this scheme is not affected by our attack. However, this transformation comes with an additional overhead in the ciphertext length as well as a non negligable overhead due to the fact that a practical implementation of the Fujisaki-Okamoto construction requires to implement at least one secure hash function. Following [6], the best known SHA-1 (resp. SHA-256) implementation requires approximately 8.000 GE (resp. 11.000 GE) which, based on the estimation in [3], would almost double the hardware implementation cost for TCHo . Moreover, this transformation is only efficient in the case where instead of using a truly random number generator for the encryption of TCHo a pseudo random number generator is used, which further increases the hardware complexity. Our result implies that TCHo cannot be used without the Fujisaki-Okamoto transformation. This in turn implies that the efficiency gain for low cost hardware devices compared to well established public key crypto systems like ECC vanishes.

One the positive side, our results can also be interpreted as an indication that breaking TCHo is equivalent to solving the low weight multiple problem.

Finally, our attack is based on a new technique that can be seen as an adaptive differential attack on public key systems. We believe that this technique can be useful for the cryptanalysis of other schemes as well.

The paper is organized as follows: In Section 2 we recall the encryption and decryption procedures for TCHo . In Section 3 we present our adaptive differential attack whose running time is discussed in detail in Section 4.

2 The TCHo cipher

The encryption of a message using TCHo can be seen as transmitting a message over a noisy channel. Given the trapdoor, i.e. the secret key, allows to reduce the noise to a level where decoding of the encrypted message is possible.

The secret key of TCHo is a polynomial $K \in \mathbb{F}_2[x]$ of degree d. We denote its coefficients by k_0 up to k_d, i.e.

$$K = k_0 \oplus k_1 x \oplus k_2 x^2 \oplus \cdots \oplus k_d x^d.$$

For the key K it holds that $k_0 = k_d = 1$. Given the polynomial K we associate the following matrix M with ℓ columns and $\ell - d$ rows to it

$$M = \begin{pmatrix} k_0 & k_1 & \dots & k_d & 0 & 0 & \dots & 0 \\ 0 & k_0 & k_1 & \dots & k_d & 0 & \dots & 0 \\ & & \ddots & & & \ddots & & \\ 0 & 0 & \dots & 0 & k_0 & k_1 & \dots & k_d \end{pmatrix} \tag{1}$$

The weight of the secret polynomial, i.e. the number of non-zero coefficients is denoted by w_K. For TCHo this weight is small. The public key consists of a polynomial $P \in \mathbb{F}_2[x]$ whose degree is in a given interval $[d^P_{\min}, d^P_{\max}]$ and is chosen such that K is a multiple of P. The length k of the plaintext can be chosen arbitrarily, however following the proposed parameters in Table 1 we exemplarily choose the case where the plaintexts are 128 bit vectors. The length of the ciphertext is denoted by ℓ. Furthermore TCHo uses a random source with bias γ.

For simplicity of the description we assume that $\ell - d$ is divisible by 128. Denote $N = \frac{\ell-d}{128}$. The attack has an identical complexity in the general case as can be seen from the experimental results in Section 4.3.

	k	$d^P_{min} - d^P_{max}$	d	w_K	γ	ℓ
I65	128	$5800 - 7000$	25820	45	0.981	50000
II65	128	$8500 - 12470$	24730	67	0.987	68000
III	128	$3010 - 4433$	44677	25	$1 - \frac{3}{64}$	90000
IV	128	$7150 - 8000$	24500	51	0.98	56000
V	128	$6000 - 8795$	17600	81	$1 - \frac{3}{128}$	150000
VI	128	$9000 - 13200$	31500	65	$1 - \frac{1}{64}$	100000

Table 1. Set of parameters proposed for TCHo (see [3])

2.1 Encryption

TCHo encrypts a plaintext $m \in \mathbb{F}_2^{128}$ by repeating the message m contiguously and afterwards truncating it to ℓ bits. This results in a vector in $\mathbb{F}_2^{\ell}$. To this vector a random string $r \in \mathbb{F}_2^{\ell}$ is added. This random string is not balanced, but (highly) unbalanced in the sense that it contains far more zeros than ones. The bias is denoted by γ. In addition, the first ℓ bits, denoted by $p \in \mathbb{F}_2^{\ell}$, of the output of an (randomly initialized) LFSR with characteristic polynomial P is added. Thus the encryption of the message m is

$$c = R(m) \oplus r \oplus p$$

where $R(m) \in \mathbb{F}_2^{\ell}$ denotes the repeated and truncated version of m. The encryption process is shown in Figure 1.

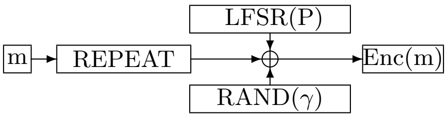

Fig. 1. Encryption with TCHo .

2.2 Decryption

Given a ciphertext $c \in \mathbb{F}_2^\ell$ decryption works as follows.

1. The ciphertext c is multiplied by M, the matrix associated with the secret polynomial K given by (1). Let $t := Mc$ where $t \in \mathbb{F}_2^{\ell-d}$. In doing so, the contribution from the LFSR with characteristic polynomial P vanishes as a result of K being divisible by P. Thus t corresponds to an encoding of the original message m xored with a random bit string of bias approximately γ^{w_K} (see [3] for details). Now, as w_K is small, this bias is still large enough to recover the message in the next step with high probability.
2. A majority logic decoding is performed on t. More precisely for each $0 \leq j < 128$ the sum
$$s_j = \sum_{i=0}^{N-1} t_{128i+j}$$
is computed over the integers. Remember that $N = \frac{\ell-d}{128}$ and note that this is exactly the position where for simplicity of the description we require N to be an integer. When this sum is greater or equal to $N/2$ the result of the decoding is 1, otherwise it is 0. The result of this majority logic decoding is a vector $e \in \mathbb{F}_2^{128}$ where
$$e_j := \begin{cases} 1 & \text{if } s_j \geq \text{N/2} \\ 0 & \text{if } s_j < \text{N/2} \end{cases} \qquad j \in \{0, \ldots, 127\}$$
3. Finally the vector e is multiplied by an invertible 128×128 bit matrix T to recover the message $m := Te$. Note that this matrix T depends on K and is therefore unknown to the attacker.

2.3 Security Considerations

Given the public key P the problem to recover the secret key K is referred to as the *Low Weight Polynomial Multiple Problem*, i.e. given a polynomial P find a polynomial K that is divisible by P, has a bounded degree and low weight. This problem is supposed to be hard. In [7] several algorithms were presented to solve this problem. Additionally, an algorithm based on lattices was presented in [2]. None of these approaches is capable to break TCHo .

As mentioned above TCHo is clearly not IND-CCA2 secure. This is due to the fact that it is trivially malleable: Given a ciphertext c for a message m then $c \oplus R(m')$ is an encryption of $m \oplus m'$. Moreover, if given a ciphertext, changing only one bit is likely to be a valid ciphertext for the same message. In [3] the authors propose to use a generic hybrid construction to obtain a hybrid scheme that offers CCA2 security. However, this can only be applied efficiently in the case where the random source is actually a pseudo random source. Using a pseudo random source will increase the hardware complexity and is therefore a suboptimal solution. Furthermore, a secure hash function has to be implemented.

Below, we present a chosen ciphertext attack that recovers the secret key nearly in linear time. This attack shows that TCHo is not even CCA1 secure. Moreover, given a decryption oracle one can efficiently recover the secret key. From a practical perspective, such an attack is by far more important than an attack based on the malleability of the scheme.

3 The Attack

Our attack strategy is to decrypt pairs of ciphertexts with a carefully chosen difference. These differences are chosen such that the intermediate states in the decryption process, after multiplication with the secret matrix M, differ in one bit if and only if a certain key bit is set. With a high probability this difference will cause a difference in the output of the decryption oracle. Thus, after a few iterations of this approach we are able to decide with overwhelming probability if a certain key bit is set or not.

The reason why a difference after multiplication with the matrix M yields a difference after the majority logic decoding with good probability lies in the fact that randomly chosen vector is likely to be balanced. In this case the one bit difference between the two states will cause two different results after the majority logic decoding and therefore two different results after the last step in the decoding procedure, i.e. after multiplication with the invertible matrix T.

To illustrate our attack, we first demonstrate how to recover k_0 using the approach outlined above. Note that $k_0 = 1$ in any case, and thus there is no need to recover it, still this will clarify our attack strategy.

The following technical lemma will be used to estimate the success probability of our attack.

Lemma 1. *For $N \geq 1$ we have*

$$\frac{1}{\sqrt{2}} \frac{1}{\sqrt{\pi(N+1)}} < \frac{\binom{N}{\lceil N/2 \rceil}}{2^N} < 2\frac{1}{\sqrt{\pi N}}.$$

Proof. For even N the bounds obtained by Stirling's approximation state

$$\frac{1}{\sqrt{2}} \frac{1}{\sqrt{\pi N}} < \frac{\binom{N}{N/2}}{2^N} < 2\frac{1}{\sqrt{\pi N}}.$$

Looking at Pascals triangle, we get for odd N that $\binom{N}{(N+1)/2} = \frac{1}{2}\binom{N+1}{(N+1)/2}$, thus

$$\frac{1}{\sqrt{2}} \frac{1}{\sqrt{\pi(N+1)}} < \frac{\binom{N}{(N+1)/2}}{2^N} = \frac{\binom{N+1}{(N+1)/2}}{2^{N+1}} < 2\frac{1}{\sqrt{\pi(N+1)}}.$$

Combining both cases we have

$$\frac{1}{\sqrt{2}} \frac{1}{\sqrt{\pi(N+1)}} < \frac{\binom{N}{\lceil N/2 \rceil}}{2^N} < 2\frac{1}{\sqrt{\pi N}}$$

□

3.1 Recovering k_0

Recovering k_0 is very simple. First, one simply decrypts a randomly chosen bitstring. In a second step the first bit of the randomly chosen bitstring is flipped and is decrypted again. If the two decrypted messages differ then k_0 has to be equal to 1. This idea is explained in detail below.

Algorithm 1 *(Recover k_0)*

1. *Choose a random vector $c \in \mathbb{F}_2^\ell$ and let it be decrypted by the oracle. Let its decryption be m.*
2. *Compute the vector $c' = c \oplus \delta$ where $\delta = (1, 0, \dots, 0)$, i.e. flip the first bit of the ciphertext. Let c' be decrypted by the oracle and denote its decryption be m'.*
3. *If $m \neq m'$ we deduce that $k_0 = 1$.*
4. *Repeat these steps with a new random vector.*
5. *If after α repetitions no difference occurred, we deduce that $k_0 = 0$.*

The difference of the intermediate states, $t = Mc$ and $t' = Mc'$, after the first step in the decryption process (see Section 2.2) is

$$\Delta = M(c \oplus c') = M(1, 0, \dots, 0)^t = (k_0, 0, \dots, 0)^t.$$

Therefore t and t' differ if and only if k_0 is 1. Note that the attacker has no access to these values. However, if by coincidence this one bit difference causes a difference in the result of the majority logic decision during decryption, a difference will occur in the decrypted messages visible to the attacker. Let us denote by s_0 the value corresponding to the sum in the majority logic decoding step for t and s'_0 be the corresponding value for t', i.e.

$$s_0 = \sum_{i=0}^{N-1} t_{128i}, \quad s'_0 = \sum_{i=0}^{N-1} t'_{128i}$$

If s_0 is $\lceil N/2 - 1 \rceil$ (resp. $\lceil N/2 \rceil$) and the value of s'_0 is $\lceil N/2 \rceil$ (resp. $\lceil N/2 - 1 \rceil$) the values of e and e' after the majority logic decoding will differ in their first coordinates as

$$e_0 := \begin{cases} 1 & \text{if } s_0 \geq \text{N/2} \\ 0 & \text{if } s_0 < \text{N/2} \end{cases}$$

and

$$e'_0 := \begin{cases} 1 & \text{if } s'_0 \geq \text{N/2} \\ 0 & \text{if } s'_0 < \text{N/2} \end{cases}.$$

Therefore, in this case we can conclude that $k_0 = 1$ and moreover get

$$m \oplus m' = T(e \oplus e') = T(1, 0, \dots, 0)^t,$$

where T is the key dependent matrix used in the last step of the decryption procedure. The key point for the running time of the attack is that this happens with a non negligible probability. The fact that c was chosen randomly and M has maximal rank implies that t is a random vector in $\mathbb{F}_2^{\ell-d}$. Remember that s_0 equals $N/2$ if and only if exactly half of the bits t_{128j}, $j \in \{0, \ldots, N-1\}$ are one and the other half is zero.

Thus, applying Lemma 1, we get

$$\mathcal{P}(s_0 = \lceil N/2 \rceil) = \frac{\binom{N}{\lceil N/2 \rceil}}{2^N} > \sqrt{\frac{1}{2\pi(N+1)}}.$$

Therefore, the probability to get a difference in m and m' is given by

$$\begin{aligned}\mathcal{P}(m \neq m' \mid k_0 = 1) &= \mathcal{P}(s_0 = \lceil N/2 \rceil \text{ and } s_0' = \lceil N/2 - 1 \rceil) \\ &\quad + \mathcal{P}(s_0 = \lceil N/2 - 1 \rceil \text{ and } s_0' = \lceil N/2 \rceil) \\ &= \frac{\binom{N}{\lceil N/2 \rceil}}{2^N} \mathcal{P}(t_0 = 0) + \frac{\binom{N}{\lceil N/2-1 \rceil}}{2^N} \mathcal{P}(t_0 = 1) \\ &> \sqrt{\frac{1}{8\pi(N+1)}}.\end{aligned}$$

Next we consider the error probability, i.e. the probability that, after running Algorithm 1 we deduce $k_0 = 0$ while it holds that $k_0 = 1$. This is given by the probability that, under the condition $k_0 = 1$, in none of the α tries a difference occurred. This probability can be upper bounded by

$$\mathcal{P}(\text{ error }) \leq \left(1 - \sqrt{\frac{1}{8\pi(N+1)}}\right)^{\alpha}$$

which is exponentially small in α. Note furthermore that following Algorithm 1 we will never erroneously deduce $k_0 = 1$ while it holds that $k_0 = 0$.

Example 1. For the parameter set (IV) in Table 1 we get for $\alpha = 100$ tries an error probability less than 0.006 and for $\alpha = 200$ an error probability less than 2^{-14}.

3.2 Recovering all key bits

The method to recover other key bits than k_0 generalizes the idea outlined above. Our goal is to construct two ciphertexts with a certain difference, such that with a high probability the majority decision flips one bit of the decrypted message if the keybit we are looking for is set. Below, we consider only the case where the attacker wants to recover k_n where $n < \ell - d$. This is the most challenging – and for all but the first proposed parameters the only– case that occurs. In the case where $n \geq \ell - d$ one can easily adopt the ideas described below to recover k_n with two decryptions only.

Choosing the difference. Let us assume that we already successfully recovered the bits k_0 up to k_{n-1} and want to recover k_n next. Denote the vector after multiplying the difference $\delta \in \mathbb{F}_2^\ell$ with the secret matrix M by Δ.

$$\Delta = \begin{pmatrix} k_0 & k_1 & \dots & k_d & 0 & 0 & \dots & 0 \\ 0 & k_0 & k_1 & \dots & k_d & 0 & \dots & 0 \\ & & \ddots & & & \ddots & & \\ 0 & 0 & \dots & 0 & k_0 & k_1 & \dots & k_d \end{pmatrix} \begin{pmatrix} \delta_0 \\ \delta_1 \\ \vdots \\ \delta_\ell \end{pmatrix}$$

We wish to have the difference $\Delta = (1 \oplus k_n, 0, \dots, 0, 1, 0, \dots, 0)^t$ where the 1 is in the $(n+1)$-th position.

For the attack we have to distinguish two cases. First consider the case where n is not divisible by 128. In this case, for two ciphertexts c and c' with the difference δ the values s_0 and s_0' will differ if k_n is not set. In the case where n is divisible by 128 the bit in the $(n+1)$-th position will contribute to the same sum s_0. In order to avoid a cancelation of these contributions special care has to be taken in the choice of the ciphertexts.

Given the knowledge about the key we already have we can compute a vector $\delta' \in \mathbb{F}_2^n$ such that

$$M'\delta' = (0, \dots, 1)^t \tag{2}$$

where M' is the following $n \times n$ sub block of M:

$$M' = \begin{pmatrix} k_0 & k_1 & \dots & k_{n-1} \\ 0 & k_0 & \dots & k_{n-2} \\ 0 & & \ddots & \\ 0 & \dots & 0 & k_0 \end{pmatrix}$$

Note that, as $k_0 = 1$ the matrix M' is bijective and thus the existence of δ' fulfilling (2) is guaranteed.

Now we can construct $\delta \in \mathbb{F}_2^\ell$ the following way: The first entry, δ_0, is computed as shown below, then the vector δ' is appended and finally δ is filled up with zeros, i.e.

$$\begin{array}{lcl} \delta_0 = \sum_{i=0}^{n-2} \delta_i' k_{i+1} \oplus 1 & & \\ \delta_i = \quad \delta_{i-1}' & & \text{for } 1 \le i \le n \\ \delta_i = \quad 0 & & \text{for } n+1 \le i < \ell. \end{array} \tag{3}$$

One verifies

$$\begin{aligned} \Delta = M\delta &= \left(k_0 (\sum_{i=0}^{n-2} \delta_i' k_{i+1} \oplus 1) \oplus \sum_{i=0}^{n-1} \delta_i' k_{i+1}, 0 \dots, 0, 1, 0 \dots, 0 \right)^t \\ &= \left(k_0 \oplus \delta_{n-1}' k_n, 0 \dots, 0, 1, 0 \dots, 0 \right)^t \\ &= (1 \oplus k_n, 0 \dots, 0, 1, 0, \dots, 0)^t, \end{aligned}$$

where the last equality follows as (2) implies $\delta_{n-1}' = 1$.

Choosing the ciphertext. Unlike in the case where we wanted to recover k_0 we will not use arbitrary random ciphertexts in this case, but restrict ourselves to certain types of vectors. As mentioned above, this will ensure that we can handle the case where we want to recover bits k_n where n is divisible by 128. Moreover, given the knowledge about the key we already have, carefully choosing the ciphertext will improve the probability of obtaining pairs such that a difference after multiplying with the secret matrix M will cause a difference at the output of the decryption oracle.

We choose the ciphertext c such that we obtain a vector $t = Mc$ with the properties that 1.) it starts with repeated blocks of 256 bits, where the first bit is one and the remaining 255 bits are zero, these repeated blocks are truncated to give an n bit string, 2.) has a zero in the $(n+1)$-th position, 3.) the remaining bits are randomly distributed. Since we know the key bits k_0 to k_{n-1}, we are able to compute the first part $\hat{c} \in \mathbb{F}_2^n$ of the ciphertext the same way we computed δ'. More precisely, using the matrix M' defined above, we are going to compute $\hat{c}$ such that

$$M'\hat{c} = b$$

where b consist of repeated blocks of 256 bits of the form $(1, 0^{(255)})$, i.e.

$$b = (1, 0^{(255)}, 1, 0^{(255)}, \dots) \in \mathbb{F}_2^n .$$

To get the zero entry at the $(n+1)$-th position of t, we will set $d+1$ consecutive bits of c equal to zero and finally the remaining bits of c are chosen uniformly at random. The ciphertext then has the structure

$$c = (\hat{c}, 0^{(d+1)}, r) \in \mathbb{F}_2^\ell \tag{4}$$

where $r \in \mathbb{F}_2^{\ell-n-(d+1)}$ is randomly chosen.

The one at the first position together with the zero at the $(n+1)$-th position of t will ensure that we can handle the case n divisible by 128. The repeated blocks of a one followed by 255 zeros at the beginning of t will increase the probability to get a difference in the decryptions provided that k_n is zero. This is explained in detail in Section 4.

Algorithm 2 *(Recover k_n)*

1. *Choose a random vector $r \in \mathbb{F}_2^{\ell-n-d-1}$ and compute c as described by (4). Let c be decrypted by the oracle. Let its decryption be m.*
2. *Compute the vector $c' = c \oplus \delta$ where δ was computed following (3) and let it be decrypted by the oracle. Let its decryption be m'.*
3. *If $m \oplus m' = T(1, 0, \dots, 0)^t$ we deduce that:*
   (a) $k_n = 0$ in the case where $n \neq 0 \bmod 128$
   (b) $k_n = 1$ in the case where $n = 0 \bmod 128$
4. *Repeat the steps with a new random vector.*
5. *If after α repetitions no difference equal to $T(1, 0, \dots, 0)^t$ occurred, we deduce that*
   (a) $k_n = 1$ in the case where $n \neq 0 \bmod 128$

(b) $k_n = 0$ *in the case where* $n = 0 \bmod 128$

Note that, after performing the algorithm to recover k_0 the value $T(1, 0, \dots, 0)^t$ is known to the attacker.

4 Analysis of the Attack

We now analyze the success probability and running time of Algorithm 2. We distinguish two cases depending on $n \bmod 128$.

4.1 $n \neq 0 \bmod 128$

In this case the vectors $t = Mc$ and $t' = Mc'$ differ by

$$\Delta = t \oplus t = M(c \oplus c') = M\delta,$$

where

$$\Delta = (1 \oplus k_n, 0, \dots, 0, 1, 0, \dots, 0)^t$$

i.e. the vectors differ in their first coordinate if and only if $k_n = 0$ and in their $(n+1)$-th coordinate. Assume that $k_n = 0$. Due to (4) it holds that $t_0 = 1$ and $t'_0 = 0$ and thus the sums used for the majority logic decoding step are related by $s_0 = s'_0 + 1$. Analogously we have $s_n + 1 = s'_n$

Now, lets assume that the vector t is such that the sum used for the majority logic decoding step is $s_0 = \lceil N/2 \rceil$ (and thus $s'_0 = \lceil N/2 - 1 \rceil$). If this happens and additionally s_n and s'_n are either both less than $\lceil N/2 \rceil$ or both greater or equal to $\lceil N/2 \rceil$ then the corresponding vectors after the majority logic decoding differ in their first coordinate exactly. Thus Algorithm 2 will successfully deduce $k_n = 0$. Note that, due to the relation $s_n + 1 = s'_n$ the condition that either both values s_n and s'_n are smaller or both greater or equal to $\lceil N/2 \rceil$ is equivalent to $s'_n \neq \lceil N/2 \rceil$.

Remember that $t = Mc$ is of the form

$$t = (b, 0, r)$$

where $b \in \mathbb{F}_2^n$ is a vector consisting of repeated blocks of 256 bits, where the first bit is one and the remaining 255 bits are zero, and $r \in \mathbb{F}_2^{\ell-d-1-n}$ is a randomly chosen vector. Considering the bits t_{128j} that contribute to s_0, we see that the first $\lfloor n/128 \rfloor$ bits are balanced. Thus, $s_0 = \lceil N/2 \rceil$ if and only if half of the bits of r contributing to s_0 equal zero and the other half equals one. Therefore, the probability that $s_0 = \lceil N/2 \rceil$ equals

$$\mathcal{P}(s_0 = \lceil N/2 \rceil) = \frac{\binom{N'}{\lceil N'/2 \rceil}}{2^{N'}},$$

where $N' = \lceil \frac{\ell-d-1-n}{128} \rceil$. As n is not divisible by 128 the first $\lfloor n/128 \rfloor$ bits contributing to s'_n are all zero. Thus

$$\mathcal{P}(s'_n = \lceil N/2 \rceil) = \frac{\binom{N'}{\lceil N/2 \rceil}}{2^{N'}}.$$

It follows that the success probability

$$p := \mathcal{P}(m \oplus m' = T(1,0,\dots,0)^t \mid k_n = 0)$$

can be upper bounded by

$$\begin{aligned} p &= \mathcal{P}(s_0 = \lceil N/2 \rceil)\, \mathcal{P}(s'_n \neq \lceil N/2 \rceil) \\ &= \frac{\binom{N'}{\lceil N'/2 \rceil}}{2^{N'}} \left(1 - \frac{\binom{N'}{\lceil N/2 \rceil}}{2^{N'}}\right) \\ &\geq \frac{\binom{N'}{\lceil N'/2 \rceil}}{2^{N'}} \left(1 - \frac{\binom{N}{\lceil N/2 \rceil}}{2^{N}}\right) \\ &> \sqrt{\frac{1}{2\pi(N'+1)}} \left(1 - 2\sqrt{\frac{1}{\pi N}}\right) . \end{aligned}$$

Next, let us consider the probability that, after running Algorithm 2 we deduce $k_n = 1$ while it holds that $k_n = 0$. This is given by the probability that, under the condition that $k_n = 0$, in none of the α tries a difference equal to $T(1,0,\dots,0)$ occurred. It can be upper bounded by

$$\mathcal{P}(\text{ error }) \leq (1-p)^{\alpha}$$

which is exponentially small in α. Note that in the case where $k_n = 0$ the expected running time is $1/p$. As the weight of K is small, this is the running time for most of the cases.

4.2 $n = 0 \bmod 128$

Like before the vectors $t = Mc$ and $t' = Mc'$ differ by

$$\varDelta = t \oplus t = M(c \oplus c') = M\delta,$$

i.e. the vectors differ in their first coordinate if and only if $k_n = 0$ and in their $(n+1)$-th coordinate. Due to (4) we have $t_0 = 1$ and $t'_0 = k_n$ and $t_n = 0$ and $t'_n = 1$. Now, as n is divisible by 128, the first and the $(n+1)$-th coordinate both contribute to the value of s_0 (resp. s'_0). Hence, we get $s'_0 = s_0 + k_n$.

Now if $k_n = 1$ and $s_0 = \lceil N/2 - 1 \rceil$ (and thus $s'_0 = \lceil N/2 \rceil$) the corresponding vectors e and e' after the majority logic decoding differ in their first coordinate exactly. Thus Algorithm 2 will successfully deduce $k_n = 1$.

Again the special choice of c ensures that the first $\lfloor n/128 \rfloor$ bits of t'_{128j} are balanced. Therefore the probability of s'_0 being $\lceil N/2 \rceil$ can be upper bounded by

$$\begin{aligned} \mathcal{P}(m \oplus m' = T(1,0,\dots,0)^t \mid k_n = 1) = \mathcal{P}(s'_0 = \lceil N/2 \rceil) \\ &= \frac{\binom{N'}{\lceil N'/2 \rceil}}{2^{N'}} \\ &> \sqrt{\frac{1}{2\pi(N'+1)}} \end{aligned}$$

where again $N' = \lceil \frac{\ell-d-1-n}{128} \rceil$. Finally, let us consider the probability that, after running Algorithm 2 we deduce $k_n = 0$ while it holds that $k_n = 1$. This is given by the probability that, under the condition that $k_n = 1$, in none of the α tries a difference equal to $T(1,0,\ldots,0)^t$ occurred. It can be upper bounded by

$$\mathcal{P}(\text{ error }) \leq \left(1 - \sqrt{\frac{1}{2\pi(N'+1)}}\right)^{\alpha}$$

which is exponentially small in α.

4.3 Experimental Results

We implemented the described attack against TCHo in C/C++ using Shoup's NTL library. We were able to derive the secret key for each proposed parameter set of [3] on a Core2 Duo 2.2 GHz laptop in less than 20 hours. The individual timings are given in Table 2.

	k	d	w_K	ℓ	time in h
I65	128	25820	45	50000	2
II65	128	24730	67	68000	4.5
III	128	44677	25	90000	7
IV	128	24500	51	56000	3
V	128	17600	81	150000	20
VI	128	31500	65	100000	13

Table 2. Time to recover the secret key

One implementation detail that is worth mentioning, is the computation of δ' (resp. $\hat{c}$). A straightforward approach might be to compute the inverse of the matrix M' of known bits. This has however an utterly bad performance, so that it is not even possible to consider matrices of dimension 5000, which is a rather small example compared to the size of secret polynomial. The best idea to compute δ' and $\hat{c}$ is to solve the corresponding system of equations. We started by using the method provided by NTL, but its performance was still unsatisfactory. Because of the extreme sparsity of the matrix M' and the additionally a priori given triangular form, it is obvious that solving such a system of equations over $\mathbb{F}_2$ should not require much computation resources. Therefore we implemented a simple backwards substitution using an array to store the known one-bits and a obtained very efficient method to compute the required values δ' and $\hat{c}$.

The value of α can be chosen rather large to get the probability of an error close to zero, since the expected number of encryptions to find a zero-bit does not depend on α and the number of one-bits is very small compared to the size of the key.

Also notice that it is possible to run an arbitrary number of instances in parallel to find the correct differences at the end of the decryption process.

There are several possibilities for further improvements of the actual attack. One could guess blocks of zeros, make use of the ability to detect missing onekeybits or reuse *good* ciphertext pairs. These improvements would allow to speed up the attack by some (small) factors.

Acknowledgement

We like to thank the authors of [3] for providing us there sample implementation of TCHo as well as for helpful comments about the cipher.

References

1. Masayuki Abe, Rosario Gennaro, Kaoru Kurosawa, and Victor Shoup. Tag-KEM/DEM: A new framework for hybrid encryption and a new analysis of Kurosawa-Desmedt KEM. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 128–146. Springer, 2005.
2. Laila El Aimani and Joachim von zur Gathen. Finding low weight polynomial multiples using lattices. Cryptology ePrint Archive, Report 2007/423, 2007. http://eprint.iacr.org/.
3. Jean-Philippe Aumasson, Matthieu Finiasz, Willi Meier, and Serge Vaudenay. TCHo : A hardware-oriented trapdoor cipher. In Josef Pieprzyk, Hossein Ghodosi, and Ed Dawson, editors, *ACISP*, volume 4586 of *Lecture Notes in Computer Science*, pages 184–199. Springer, 2007.
4. Richard P. Brent and Paul Zimmermann. Algorithms for finding almost irreducible and almost primitive trinomials. In *The Fields Institute, Toronto*, page 212, 2003.
5. Anne Canteaut and Florent Chabaud. A new algorithm for finding minimum-weight words in a linear code: Application to McEliece's cryptosystem and to narrow-sense BCH codes of length 511. *IEEE Transactions on Information Theory*, 44(1):367–378, 1998.
6. Martin Feldhofer and Christian Rechberger. A case against currently used hash functions in RFID protocols. In Robert Meersman, Zahir Tari, and Pilar Herrero, editors, *OTM Workshops (1)*, volume 4277 of *Lecture Notes in Computer Science*, pages 372–381. Springer, 2006.
7. Matthieu Finiasz and Serge Vaudenay. When stream cipher analysis meets public-key cryptography. In Eli Biham and Amr M. Youssef, editors, *Selected Areas in Cryptography*, volume 4356 of *Lecture Notes in Computer Science*, pages 266–284. Springer, 2006.
8. Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 537–554. Springer, 1999.
9. Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *IEEE Symposium on Foundations of Computer Science*, pages 124–134, 1994.