Practical (Second) Preimage Attacks on TCS_SHA-3

Gautham Sekar[1] and Soumyadeep Bhattacharya[2]

[1] Indian Statistical Institute, Chennai Centre,
SETS Campus, MGR Knowledge City, CIT Campus, Taramani,
Chennai 600113, India.
[2] Institute of Mathematical Sciences,
CIT Campus, Taramani,
Chennai 600113, India.
sgautham@isichennai.res.in,sbhtta@imsc.res.in

Abstract. TCS_SHA-3 is a family of four cryptographic hash functions that are covered by an US patent (US 2009/0262925). The digest sizes are 224, 256, 384 and 512 bits. The hash functions use bijective functions in place of the standard, compression functions. In this paper we describe first and second preimage attacks on the full hash functions. The second preimage attack requires negligible time and the first preimage attack requires $O(2^{36})$ time. In addition to these attacks, we also present a negligible-time second preimage attack on a strengthened variant of the TCS_SHA-3. All the attacks have negligible memory requirements.

Keywords: Cryptanalysis, hash function, (second) preimage attack

1 Introduction

A hash function H takes an arbitrary length bit string M as input and outputs a fixed length bit string h (called *hash value* or *digest*). A cryptographic hash function is meant to satisfy certain security properties, the most important of which are listed below.

- **First preimage resistance**: given h, it is computationally infeasible to find an M such that $H(M) = h$.
- **Second preimage resistance**: given an M and $H(M)$, it is computationally infeasible to find an $M' \neq M$ such that $H(M) = H(M')$.
- **Collision resistance**: it is computationally infeasible to find an M and an M', with $M' \neq M$, such that $H(M) = H(M')$.

The general model for cryptographic hash functions involves what is called a compression function. The function transforms a fixed-length bit string into a shorter, fixed-length bit string. The input message of a hash function, that is of arbitrary length, is partitioned into blocks of a fixed length (called the *block length*). But before this could be done, it is required that the length of the message is a multiple of the block length. Given this and some security considerations, the message is 'padded' with bits in one of several ways (some padding schemes can be found in [6, Sect. 2.4.1]). The message blocks are sequentially processed, with the compression function acting on the message blocks until all the blocks are processed. The end result is output as the digest. The general model for describing hash functions can be found in greater detail in [6, Sect. 2.4.1].

A cryptographic hash function family is proposed by Vijayarangan of the Tata Consultancy Services (hereinafter called "TCS") in [3]. The family comprises four hash functions, as four digest sizes – 224, 256, 384 and 512 bits – are allowed. In [3], the hash functions are all actually called SHA-3, except in one or few instances (see e.g. Clause 0095 of [3], where a member hash function is called TCS_SHA-3). However, as the name SHA-3 has been in use by the NIST [4], we use the less common 'TCS_SHA-3' to denote the SHA-3 of [3]. Further, we denote by TCS_SHA-3-d the member that produces d-bit digests.

The design of TCS_SHA-3 deviates from the general model in that the compression function is replaced by a bijective function. This function uses a linear feedback shift register (LFSR) and a T-function. The design goals, as stated in [3], are to "prevent hash collisions" and to "provide a secure hash function". This paper establishes that the design goals are not met.

Motivation behind this work: The TCS, headquartered in India, is among the largest IT services providers in the world, with an annual revenue of more than \$10 billion for 2011–2012 [8]. In May 2012, the company was named the fourth most valuable IT services brand worldwide based on image, reputation and *intellectual property* assessments [10]. The company's annual research report for 2008–2009 mentions the following [9]:

> In the current year, major work has been done on cryptographic algorithms and hash functions which form the basis of all data security today. Past research products from [the E-Security group of the TCS Innovation Labs, Hyderabad, India,] ... are in active use around the country [(India)] by various customers in the banking and financial services industry. Organisations using our technology, directly or indirectly, include the RBI [(Reserve Bank of India)], National Securities Depositories Limited (NSDL), Ministry of Company Affairs (MCA), and many public sector banks.

Since TCS_SHA-3 is a product of the above mentioned E-Security group of the TCS Innovation Labs (see [9]), there appears to be sufficient motivation to evaluate the security of the hash function family.

Contributions of this paper: There are three contributions. First, we report a second preimage attack that requires negligible time and negligible memory for nearly guaranteed success. Second, we describe a first preimage attack on the TCS_SHA-3-d that requires $O(2^{27} \cdot d)$ time and negligible memory. Third, we present a second preimage attack, which also requires negligible time and negligible memory for nearly guaranteed success, on a strengthened variant of the TCS_SHA-3. To the best of our knowledge, there is no prior published attack on the (strengthened) TCS_SHA-3.

Organisation of this paper: Section 2 describes the TCS_SHA-3 family of hash functions. A second preimage attack and the first preimage attack are respectively described in Sects. 3 and 4. In Sect. 5, we present the second preimage attack on the strengthened TCS_SHA-3. We conclude in Sect. 6. Appendix A provides the results of our simulations of the first preimage attack.

2 Specifications

We first list the notation and conventions, followed in the rest of this paper, in Table 1.

Table 1. Notation and conventions

Symbol/notation	Meaning
LSB	least significant bit
MSB	most significant bit
$\Gamma_i(\omega)$	ith 32-bit word ($i = 0$ denotes the least significant word) of d-bit ω
$\lvert x \rvert$	length, in bits, of x
$x_{(i)}$	ith bit ($i = 0$ denotes the LSB) of x
$x \| y$	concatenation of two 32-bit words, x and y
$\oplus$	exclusive OR

The TCS_SHA-3-d takes as input a message M of arbitrary length and returns a digest h (also denoted TCS_SHA-3-$d(M)$) of size d bits after the following sequence of processes.

1. *Padding*: The input M is partitioned into $k = \lceil |M|/d \rceil$ blocks (denoted M_1, $\ldots$, M_k), each of length d bits. Clause 0068 of [3] states that an initialization vector (IV) of length d is added to a *message* if and only if the *size of the message* is less than d. The designer should have evidently meant the following: if and only if $|M_k| < d$, for any $k \geq 1$, then M_k is added with the IV. Otherwise, the TCS_SHA-3 can only be applied to single block messages or multiple block messages each of which satisfies the condition $d \| |M|$. For the simulations of Appendix A, we make the assignments IV $= 1_2 \| \{0_2\}^{d-1}$, $d = 224, 256, 384, 512$. The assignments are motivated by a case provided in [3, Clause 0068] where the IV is chosen to be $1_2 \| \{0_2\}^{223}$ when the size of the message is less than 224 bits. In our simulations as well as the aforesaid

example case, the IV is *XORed* with the corresponding message block. In summary, the padding rule is defined as follows: for any $k \geq 1$,

$$M_k \to M_k^* := \begin{cases} M_k \oplus \mathrm{IV} & \text{if } |M_k| < d \ , \\ M_k & \text{if } |M_k| = d \ . \end{cases}$$

An implicit assumption in the above discussion is that $|M_k|$ is nonzero. Further, when $|M_k| = d$, we infer that there is no extra 'padding block' that is appended to M. This is because, in such a case, there is no message block to which the IV ($= 1_2 \| \{0_2\}^{d-1}$) could be "added".

2. *Round 1*: The first *round* has k *steps*; the steps are as follows:
   (a) *Step 1 when* $k > 1$: An arbitrarily chosen d'-bit (such that $d' \leq d$) constant c is XORed with M_1. The output, $c \oplus M_1$, is input to a bijective function F (defined later in this Sect.). Thus, a d-bit string, $F(c \oplus M_1)$, is output.
   Step 1 when $k = 1$: The arbitrarily chosen constant c is XORed with M_1^*. The output, $c \oplus M_1^*$, is input to F. Thus, a d-bit string, $F(c \oplus M_1^*)$, is output.
   (b) *Steps 2–k (i.e., when* $k \geq 2$*)*: Step i, $2 \leq i \leq k-1$, is given by the following recursion:

   $$z_i^1 = F(z_{i-1}^1 \oplus M_i) \ , \ \ z_1^1 := F(c \oplus M_1) \ ;$$

   z_ℓ^1 denotes the output of step ℓ, $1 \leq \ell \leq k-1$, of round 1.
   Step k is given by:

   $$z_k^1 = F(z_{k-1}^1 \oplus M_k^*) \ ,$$

   where z_k^1 denotes the output of step k of round 1.
3. *Round 2*: Like round 1, round 2 also proceeds iteratively. The number of steps, s, is such that k "is not always the same as" s (see [3, Clause 0070]). The input to round 2 is z_k^1, the final output of round 1. The steps are as follows:
   (a) *Step 1*: The d'-bit constant c is XORed with z_k^1. The output, $c \oplus z_k^1$, is input to the bijective function F. Thereby, a d-bit string, $F(c \oplus z_k^1)$, is generated as output.
   (b) *Steps 2–s (i.e., when* $s \geq 2$*)*:[1] Step i, $2 \leq i \leq s$, is given by the following recursion:

   $$z_i^2 = F(z_{i-1}^2 \oplus z_k^1) \ , \ \ z_1^2 := F(c \oplus z_k^1) \ ;$$

   z_ℓ^2 denotes the output of step ℓ, $1 \leq \ell \leq s$, of round 2.
4. *Rounds 3–6*: These rounds are similar to round 2. The number of steps in each of these rounds is again s. The input to round j, $3 \leq j \leq 6$, is z_s^{j-1} (i.e., the final output of round $j-1$ when z_ℓ^{j-1} denotes the output of step ℓ, $1 \leq \ell \leq s$, of round $j-1$). The steps are as follows:

[1] Perhaps the only criterion that s must satisfy is $s \geq 1$; otherwise the TCS_SHA-3-d will have only one round.

(a) *Step 1*: Constant c is XORed with z_s^{j-1}. The output, $c \oplus z_s^{j-1}$, is input to the bijective function F. Thereby, a d-bit string, $F(c \oplus z_s^{j-1})$, is generated

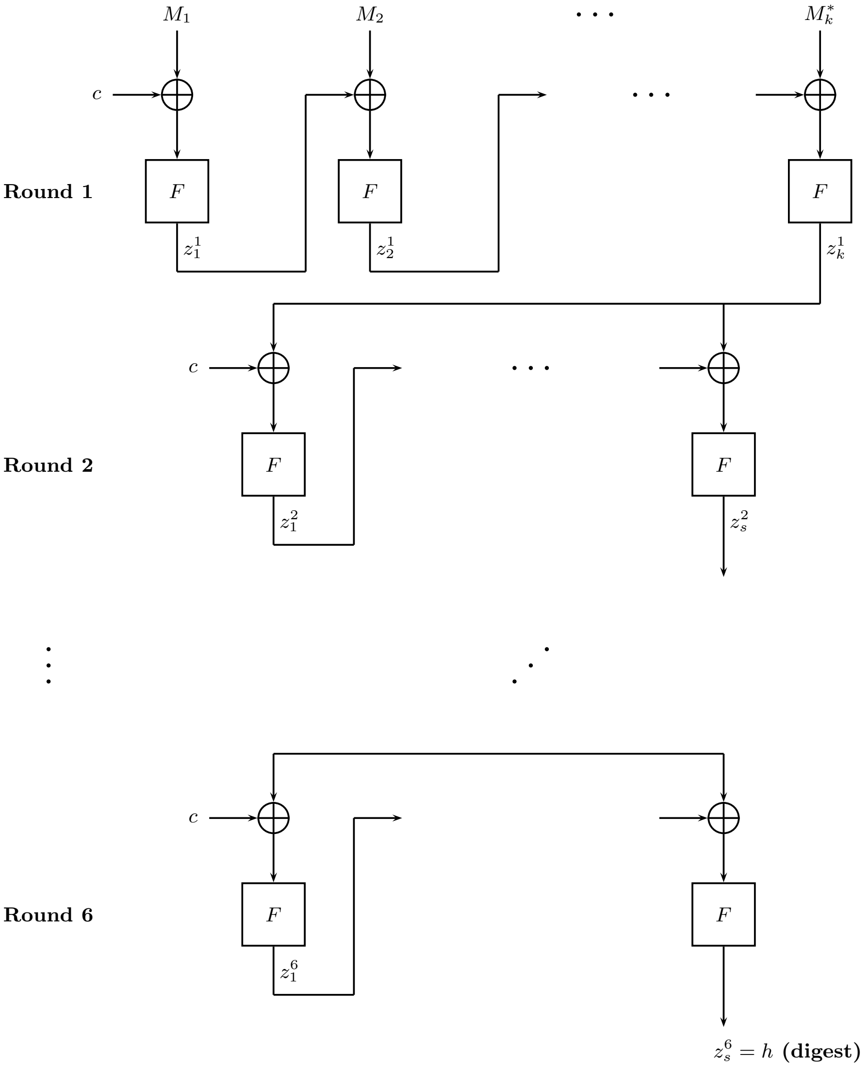

Fig. 1. The working of TCS_SHA-3 ($k \geq 2$)

[2] The (second) preimage attacks that we report in this paper are independent of the value of s due to reasons that would be understood from Sects. 3 and 4.

Algorithm 1 The bijective function $F : \{0_2, 1_2\}^d \to \{0_2, 1_2\}^d$

```
Require: d-bit input $\alpha$
Ensure: d-bit output $\lambda$
Partition: $\alpha \to \alpha_1 \| \alpha_2 \| \ldots \| \alpha_{d/32}$ such that $|\alpha_i| = 32$ for all $1 \le i \le d/32$;
Shuffle: $\alpha_i \to \beta_i$, for all $1 \le i \le d/32$, such that $\beta_{i(j)} = \alpha_{i(j/2)}$ if $2|j$ and $\beta_{i(j)} = \alpha_{i(16+(j-1)/2)}$ otherwise;
Apply T-function: $\beta_i \to \gamma_i \coloneqq 2\beta_i^2 + \beta_i \bmod 2^{32}$, for all $1 \le i \le d/32$;
Apply LFSR: $\gamma_i \to \lambda_i$, for all $1 \le i \le d/32$, such that $|\lambda_i| = 32$;[3]
Concatenate: $\lambda \coloneqq \lambda_1 \| \lambda_2 \| \ldots \| \lambda_{d/32}$;
```

Note: The TCS_SHA-3 may be strengthened by introducing cipher block chaining in Algorithm 1. This point is further explained in Sect. 5.

3 Second Preimage Attack on TCS_SHA-3

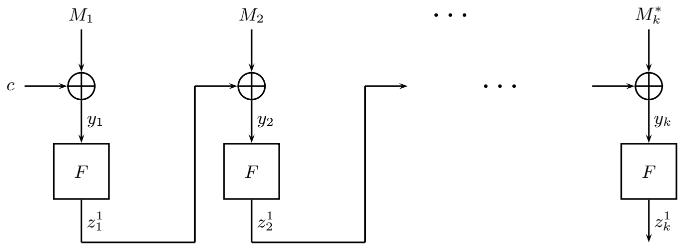

Fig. 2. Round 1 of TCS_SHA-3

Let $M = M_1 \| M_2 \| M_3 \| \cdots \| M_k^*$, $k \ge 2$ (when $k = 2$, $M = M_1 \| M_2^*$; in general, when $k = \ell > 2$, $M = M_1 \| M_2 \| \cdots \| M_\ell^*$), denote the given message and h its hash value. Let y_i denote the input to the ith invocation of the function F (see Figure 2). Let $M' = M_1' \| M_2' \| M_3 \| \cdots \| M_k^*$, with $M_1' \ne M_1$ and $M_2' \ne M_2$, denote another message, h' its hash value, and y_i' the corresponding input to the ith invocation of F. It immediately follows from Figure 2 that if $y_2' = y_2$, then the outputs of round 1 are identical. This, in turn, implies that $h' = h$ (see Figure 1) and we have a second preimage. The condition $y_2' = y_2$ implies that:

$$F(M_1 \oplus c) \oplus M_2 = F(M_1' \oplus c) \oplus M_2' \ . \tag{1}$$

[3] We omit the full description of the LFSR as it is elaborate and not relevant to our analysis (to be understood from Sects. 3 and 4). In [3], the LFSR is described in Clauses 0078–0083.

It is straightforward to see that the conditions:

$$\begin{aligned} M_2' &= F(M_1' \oplus c) \oplus M_1' \ , && (2) \\ M_1' &= M_2 \oplus F(M_1 \oplus c) \ , && (3) \end{aligned}$$

satisfy (1) and when $M_1' \neq M_1$, we have a second preimage.[4] Under reasonable assumptions of uniformity, the event $M_1' = M_1$ occurs with negligible probability.

4 First Preimage Attack on TCS_SHA-3

Figure 3 illustrates Algorithm 1 for the (sample) case when $d = 256$.

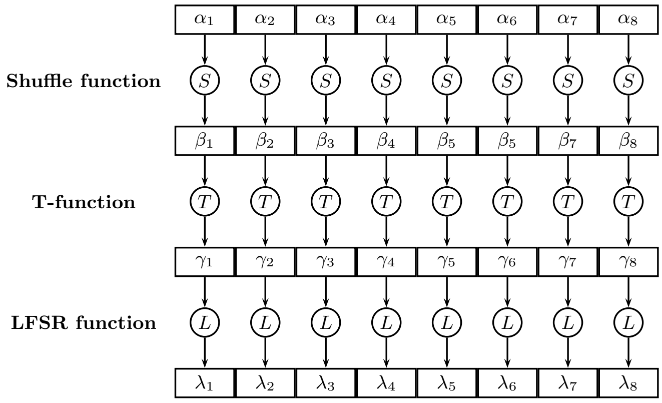

Fig. 3. The bijective function F of TCS_SHA-3-256; S, T and L are 32-bit to 32-bit functions

From Figure 3, we see that the TCS_SHA-3-256 (and the TCS_SHA-3 per se) has poor diffusion properties. A difference in α_i, for any $i \in \{1, 2, \ldots, d/32\}$, affects λ_i alone. A single-bit difference in α_i, for any $i \in \{1, 2, \ldots, d/32\}$, is ideally expected to affect 16 bits of λ_i.

Let us consider the case when $k = 1$. Then, given an input difference $\Delta\Gamma_i(M_1^*)$ for some $i \in \{1, 2, \ldots, d/32\}$, the differential characteristic is shown in Figure 4.

[4] A similar *correcting block attack* on the hash function Khichidi-1 [2] has been reported by Mouha [1, Sect. 2.6.3].

Algorithm 2 exploits this differential characteristic to recover M_1 from its corresponding digest value.

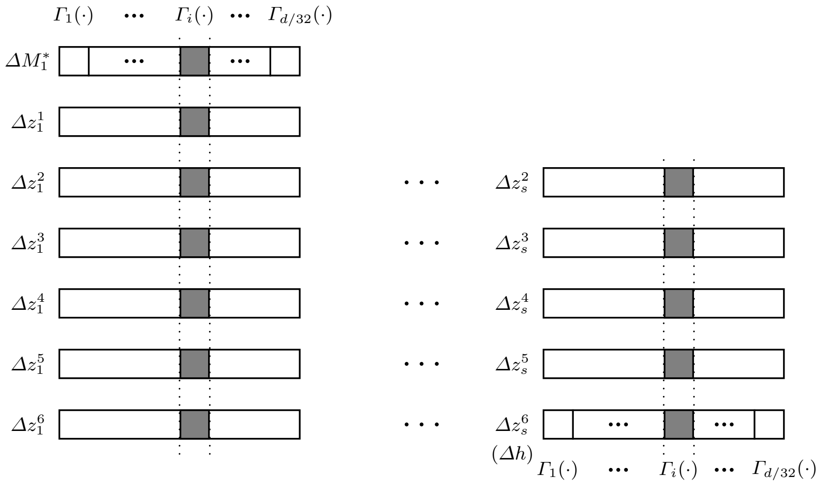

Fig. 4. Differential characteristic for TCS_SHA-3-d when $k = 1$; non-zero differences are confined to the grey boxes

Algorithm 2 requires that the attacker knows whether $|M_1| = d$ or $|M_1| < d$. However, even without this information the attacker can, at the very least, recover all but the MSB of M_1 by simply computing $M_1 = M_1^* \oplus$ IV. This is explained as follows. Suppose that $|M_1| < d$. Then, by computing $M_1 = M_1^* \oplus$ IV the attacker correctly recovers M_1. Now suppose that $|M_1| = d$. This time, the attacker is supposed to compute $M_1 = M^*$. Nevertheless, by computing $M_1 = M_1^* \oplus$ IV, the attacker is still able to correctly recover $d - 1$ LSBs of M_1 because the IV is simply $1_2 \| \{0_2\}^{d-1}$.

As we can see, Algorithm 2 has $d/32 \cdot 2^{32} = d \cdot 2^{27}$ iterations. Since $d \leq 512$, single block messages can be recovered from their respective hash values in $O(512 \cdot 2^{27}) = O(2^{36})$ time. It may therefore be extremely risky to use the TCS_SHA-3 for, say, password hashing (a well-known application of cryptographic hash functions – see [7]).

[5] In place of $\{0_2\}^{32(i-1)}$ and $\{0_2\}^{32(d/32-i)}$, one can respectively have any $32(i-1)$- and $32(d/32-1)$-bit value. We have used $\{0_2\}^{32(i-1)}$ and $\{0_2\}^{32(d/32-i)}$ for ease of understanding how the attack works.

Algorithm 2 Recovering M_1 from h when $k = 1$

```
Require: Whether $|M_1| = d$ or $|M_1| < d$
Ensure: $d$-bit output $M_1$
for $i = 1 \to d/32$ do
    for $j = \{0_2\}^{32} \to \{1_2\}^{32}$ do
        $\ell \leftarrow \{0_2\}^{32(i-1)} \| j \| \{0_2\}^{32(d/32-i)}$;[5]
        Compute $\hbar :=$ TCS_SHA-3-$d(\ell)$;
        if $\Gamma_i(\hbar) = \Gamma_i(h)$ then
            $\Gamma_i(M_1^*) \leftarrow j$;
            break;
        else
            $j \leftarrow j + 1$;
    $i \leftarrow i + 1$;
Compute $M_1^* = \Gamma_1(M_1^*) \| \Gamma_2(M_1^*) \| \ldots \| \Gamma_{d/32}(M_1^*)$;
if $|M_1| < d$ then
    Output $M_1 = M_1^* \oplus$ IV;
else
    Output $M_1 = M^*$;
```

For the case when $k > 1$, if the message blocks $M_1, \ldots, M_{k-1}$ are available to the attacker, then she may recover M_k. The attack procedure is now given by Algorithm 2 with M_1 replaced by M_k and M_1^* by M_k^*. Here again the attacker is able to recover, at the very least, all but the MSB of M_k.

5 Cipher Block Chaining in Algorithm 1: Impact on Security

In Algorithm 1, the 32-bit words α_1, α_2, ..., $\alpha_{d/32}$ are processed independently of one another (see Figure 3). This is an inference that we draw from [3] which has no explicit mention of any dependence between the processing of α_i and the processing of α_{i+1}, $1 \leq i \leq d/32 - 1$. Besides, [3] provides several implementation results but the corresponding implementation is missing. We were therefore unable to verify the correctness of the implementation results of [3]. If the implementation results are correct, then the processing of α_i and the processing of α_{i+1}, $1 \leq i \leq d/32 - 1$, may not be independent (see e.g. [3, Clauses 0094–0095]). But even in such a case, the existence of dependence must have been clearly mentioned in [3], in any of the clauses preceding Clause 0087.

When there is dependence in the form of chaining, given the structural similarities between TCS_SHA-3 and Khichidi-1 (see [2, Sect. 6]), it appears reasonable to expect the chaining mechanisms in the two cases to be identical. From [2, Sect. 6] then, we see that Figure 2 changes to Figure 5 when $k = 2$. Clearly this would also mean that FIGURE 3 of [3] is incorrect.

Note: If one goes by [2, Sect. 6], then in Figure 5, $\Gamma_0(z_1^1)$ is not XORed with $\alpha_{2,1}$, instead $\Gamma_{d/32-1}(c)$ is assigned the value of $\Gamma_0(z_1^1)$ once M_1 is processed. This may be inferred, for example, from the statement, "$H_0^{(i)} = H_7^{(i)}$" in [2, Sect. 6.1.2] (which, in fact, happens to be the only statement in the Khichidi-1 algorithms of [2] to describe the chaining process in 224-bit Khichidi-1). If, on the other hand, the statement had read, "$H_0^{(i+1)} = H_7^{(i)}$", then the chaining process would have appeared meaningful. We therefore presume that the chaining statements in the Khichidi-1 algorithms of [2] are typographically flawed in the manner described above. □

The case of the aforementioned independent processing of the 32-bit blocks, however, complies with [3, FIGURE 3] and thereby enhances our belief in the correctness of the above mentioned inference of independent processing of the α_i's ($1 \leq i \leq d/32$).

Algorithm 3 The bijective function $g : \{0_2, 1_2\}^{32} \rightarrow \{0_2, 1_2\}^{32}$

```
Require: 32-bit input $\alpha$
Ensure: 32-bit output $\lambda$
    Shuffle: $\alpha \rightarrow \beta$ such that $\beta_{(j)} = \alpha_{(j/2)}$ if $2|j$ and $\beta_{(j)} = \alpha_{(16+(j-1)/2)}$ otherwise;
    Apply T-function: $\beta \rightarrow \gamma := 2\beta^2 + \beta \bmod 2^{32}$;
    Apply LFSR: $\gamma \rightarrow \lambda$ such that $|\lambda| = 32$;
```

Let $M = M_1 \| M_2 \| M_3 \| \cdots \| M_k^*$, $k \geq 2$ (when $k = 2$, $M = M_1 \| M_2^*$; in general, when $k = \ell > 2$, $M = M_1 \| M_2 \| \cdots \| M_\ell^*$), denote the given message and h its hash value. Let $M' = M_1' \| M_2' \| M_3 \| \cdots \| M_k^*$, with $M_1' \neq M_1$ and $M_2' \neq M_2$, denote another message and h' its hash value. We now define the following for all $j \in \{1, 2, \ldots, d/32\}$:

$$\alpha_{i,j} := \begin{cases} \Gamma_{d/32-j}(M_i) & \text{when } i < k \ , \\ \Gamma_{d/32-j}(M_k^*) & \text{when } i = k \ . \end{cases}$$

Suppose that the conditions $M_1' \neq M_1$ and $M_2' \neq M_2$ are such that $\alpha_{1,i} = \alpha_{1,i}'$ for all $i \in \{1, 2, \ldots, d/32 - 1\}$ and $\alpha_{2,i} = \alpha_{2,i}'$ for all $i \in \{2, \ldots, d/32\}$. Then, from Figure 5, it follows that $h' = h$ when the following condition is satisfied:

$$g(\alpha_{1,d/32} \oplus \theta) \oplus \alpha_{2,1} = g(\alpha_{1,d/32}' \oplus \theta) \oplus \alpha_{2,1}' \ . \tag{4}$$

It is straightforward to see that the conditions:

$$\alpha_{2,1}' = g(\alpha_{1,d/32}' \oplus \theta) \oplus \alpha_{1,d/32}' \ , \tag{5}$$

$$\alpha_{1,d/32}' = \alpha_{2,1} \oplus g(\alpha_{1,d/32} \oplus \theta) \ , \tag{6}$$

satisfy (4) and when $\alpha_{1,d/32}' \neq \alpha_{1,d/32}$, we have a second preimage. Under reasonable assumptions of uniformity, the event $\alpha_{1,d/32}' = \alpha_{1,d/32}$ occurs with negligible probability.

[7] Clause 0069 of [3] states that $|c| \leq d$. If $|c| = e < d$, then $c \leftarrow \{0_2\}^{d-e} \| c$. We find no mention in [3] that it is the most significant word of c that is XORed with $\alpha_{1,1}$ – we simply make such an assumption without loss of generality.

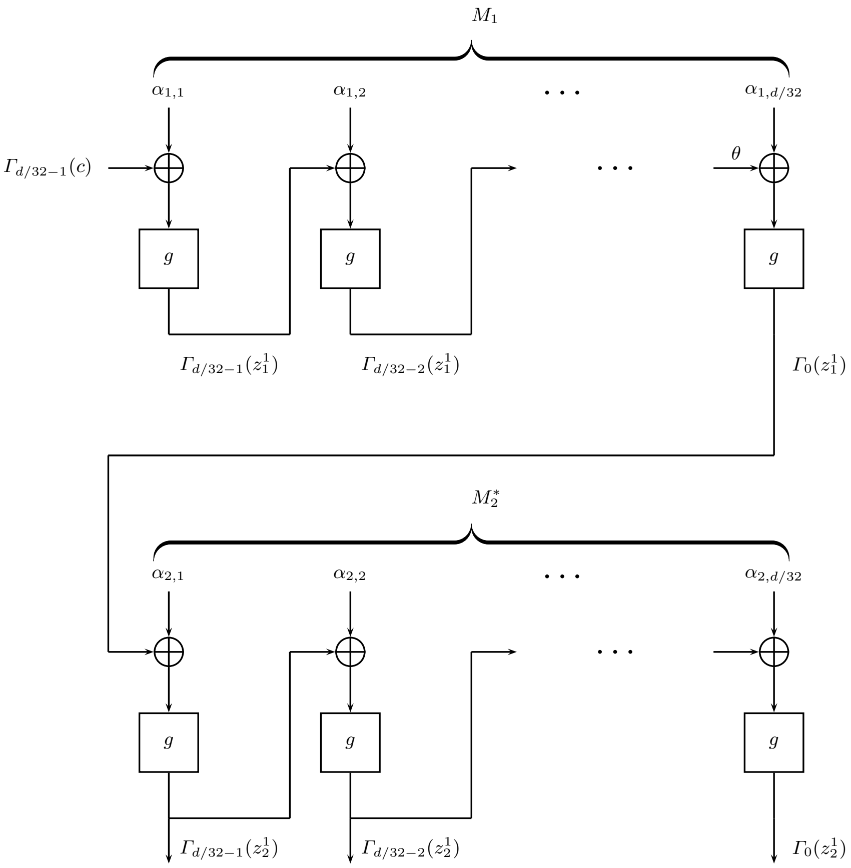

Fig. 5. Round 1 of TCS_SHA-3 ($k = 2$) with 32-bit cipher block chaining; the bijective function $g : \{0_2, 1_2\}^{32} \rightarrow \{0_2, 1_2\}^{32}$ is given by Algorithm 3; $\alpha_{1,j} = \Gamma_{d/32-j}(M_1)$ and $\alpha_{2,j} = \Gamma_{d/32-j}(M_2^*)$, for all $j \in \{1, 2, \ldots, d/32\}$ [7]

6 Conclusions and Open Problems

In this paper, we have presented practical (second) preimage attacks on the TCS_SHA-3 family of patented cryptographic hash functions. The second preimage attack requires negligible time and negligible memory for nearly guaranteed success. The attack works when the number of message blocks is at least two. The first preimage attack requires $O(2^{36})$ time and negligible memory. This attack is the most efficient (going by data requirements) on single block messages – negligible data is required in such cases. We have also reported a negligible-time/memory second preimage attack on the TCS_SHA-3 that is strengthened with 32-bit cipher block chaining. This attack also works only when the number of message blocks is at least two.

Our findings establish, among others, that the TCS_SHA-3 may be particularly unsuitable for password hashing (unless, say, it is strengthened with 32-bit cipher block chaining).

It may be an interesting exercise to find countermeasures to our attacks.

References

1. Nicky Mouha. *Automated Techniques for Hash Function and Block Cipher Cryptanalysis*. PhD thesis, Katholieke Universiteit Leuven, 2012. Available at: http://www.cosic.esat.kuleuven.be/publications/thesis-203.pdf.
2. Natarajan Vijayarangan. A NEW HASH ALGORITHM: Khichidi-1. Submission to Cryptographic Hash Algorithm Competition organised by the NIST, USA, 2008. Available at: http://ehash.iaik.tugraz.at/uploads/d/d4/Khichidi-1.pdf.
3. Natarajan Vijayarangan. Method for Designing a Secure Hash Function and a System Thereof. Patent US 2009/0262925, 22 October 2009. Available at: http://appft1.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PG01&p=1&u=/netahtml/PTO/srchnum.html&r=1&f=G&l=50&s1=20090262925.PGNR.
4. *National Institute of Standards and Technology*. Cryptographic Hash Algorithm Competition. Available at: http://csrc.nist.gov/groups/ST/hash/sha-3/index.html.
5. *National Institute of Standards and Technology*. FIRST ROUND CANDIDATES, 5 June 2012 (last updated). Khichidi-1 package available at: http://csrc.nist.gov/groups/ST/hash/sha-3/Round1/documents/Khichidi-1.zip.
6. Bart Preneel. *Analysis and Design of Cryptographic Hash Functions*. PhD thesis, Katholieke Universiteit Leuven, 1993. Available at: http://homes.esat.kuleuven.be/~preneel/phd_preneel_feb1993.pdf.
7. Bart Preneel. The State of Cryptographic Hash Functions. In Ivan Damgård, editor, *Lectures on Data Security*, volume 1561 of *Lecture Notes in Computer Science*, pages 158–182. Springer, 1998.
8. *Tata Consultancy Services*. Annual Report 2011–12. Available at: http://www.tcs.com/investors/Documents/Annual%20Reports/TCS_Annual_Report_2011-2012.pdf.
9. *Tata Consultancy Services*. Annual Research Report 2007–08. Available at: http://www.tcs.com/SiteCollectionDocuments/White%20Papers/TCS-Annual-Research-Report-2007-08.pdf.
10. *Tata Consultancy Services Press Release*. TCS recognized as Big Four IT Services brand, 07 May 2012. Available at: http://www.tcs.com/news_events/press_releases/Pages/TCS_recognized_Big_Four_IT_Services_brand.aspx.

A Experiments

We took a few sample outputs of TCS_SHA-3 and attempted to verify their first preimages. Simulations were performed on an Nvidia GeForce GT 540M GPU having 96 CUDA cores (2 multiprocessors × 48 CUDA cores/ multiprocessor) and a clock rate of 1.34 GHz. The CUDA C compiler nvcc 4.2 was used. Table 2 lists the assignments made for the simulations. This set of assignments is justified by footnotes 1–3, footnote 7, Sect. 2, Sect. 4, and the following points:

– See footnote 7. As the value of c does not affect the analysis or the experiments, we assign the value $\{0_2\}^d$ to c.
– The IV is a fixed parameter (see Sect. 2).
– The value of s is taken to be the value of s in Khichidi-1 (see [2, Sect. 6] and *khichidi.c* in the URL provided in [5]), and therefore equal to 2.
– The value of η_i, $1 \leq i \leq 6$, is taken to be the number of LFSR shifts in round i of Khichidi-1 (see [2, Sect. 6] and *khichidi.c* in the URL provided in [5]).

Table 2. Assignments for the simulations

Parameter	**Value**
c	$\{0_2\}^d$
IV	$1_2 \| \{0_2\}^{d-1}$
s	2
# LFSR shifts in round i (η_i), $1 \leq i \leq 6$	1

The worst-case system time (considering that the hashing time is maximum) taken to find the first preimage for each output is provided in Table 3.

From Sects. 2 and 4, it follows that the time/memory complexity of our first preimage attack does not change when one or more of the assignments of Table 2 are altered. Only the system time taken to find a first preimage may change.

Table 3. Sample first preimages

Digest length (in bits)	Output of compression function (in hex)	Preimage	Time taken (in seconds)
224	F18DE455 827C1EE6 00000000 00000000 00000000 00000000 00000000	“password”	28.66
256	F18DE455 827C1EE6 00000000 00000000 00000000 00000000 00000000 00000000	“password”	28.67
384	F18DE455 827C1EE6 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000	“password”	28.74
512	F18DE455 827C1EE6 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000	“password”	28.80
224	2C2B32B7 305056B5 709FF3A9 7A99955B 3D271585 78E21E7E 2848DD92	“The quick brown fox jumps”	91.28
256	2C2B32B7 305056B5 709FF3A9 7A99955B 3D271585 78E21E7E CC1779EB A3ED2089	“The quick brown fox jumps over”	103.80
384	2C2B32B7 305056B5 709FF3A9 7A99955B 3D271585 78E21E7E CC1779EB 37407107 89D12C8B EA6630B4 3929A26F 00000000	“The quick brown fox jumps over the lazy dog”	141.08
512	2C2B32B7 305056B5 709FF3A9 7A99955B 3D271585 78E21E7E CC1779EB 37407107 89D12C8B EA6630B4 3929A26F 00000000 00000000 00000000 00000000 00000000	“The quick brown fox jumps over the lazy dog”	141.14