Secure Design and Instantiation of Trapdoor-less Truncated Hash Functions

Student Name: Tarun Kumar Bansal

IIIT-D-MTech-CS-IS-11-014
July 1, 2013

Indraprastha Institute of Information Technology
New Delhi

Thesis Committee
Dr. Somitra Sanadhya (Chair)
Dr. R.K. Agrawal
Dr. Sanjit K. Kaul

Submitted in partial fulfillment of the requirements
for the Degree of M.Tech. in Computer Science,
with specialization in Information Security

©2013 Tarun Kumar Bansal
All rights reserved

Keywords: hash function and family, secure instantiation, block cipher, trapdoor, truncation, design, implementation .

Certificate

This is to certify that the thesis titled **"Secure Design and Instantiation of Trapdoor-less Truncated Hash Functions"** submitted by **Tarun Kumar Bansal** for the partial fulfilment of the requirements for the degree of *Master of Technology* in *Computer Science & Engineering* is a record of the bonafide work carried out by him under my guidance and supervision in the Security and Privacy group at Indraprastha Institute of Information Technology, Delhi. This work has not been submitted anywhere else for the reward of any other degree.

Dr. Donghoon Chang
Indraprastha Institute of Information Technology, New Delhi

Dr. Somitra Kumar Sanadhya
Indraprastha Institute of Information Technology, New Delhi

Abstract

In this work, we throw light on gaps between theoretical construction of cryptographic primitives and their practical instantiation. It is assumed that cryptographic primitives will not have any trapdoors in the given design. It is also assumed that security of a cryptographic construction should remain the same in practice as proven in theory. We show how a designer can act as adversary if he doesn't follow some basic paradigms while designing. To ensure that the basic assumptions, made while providing security proof of cryptographic construction, remain true and doubt free, we show that order of defining the underlying primitives of security protocols are important.

We use Hash-then-Truncate(HtT) construction as an example. While discussing security of HtT construction we show that order of defining hash and truncation function is important. We also show that by providing choices of hash functions from a hash family to a user, we can securely instantiate the HtT construction. In this way, we can bridge the gap between design and practice.

Acknowledgments

There are many people who helped in different ways to make this thesis possible. I would like to thank them all for their suggestions and advise throughout of my tenure Masters studies. I would like to express my deepest gratitude to my advisor Dr. Donghoon Chang for his guidance and all kinds of support. His great teaching motivated me to initiate my research in cryptography. His teaching style, creativity and extreme energy have always been a source of motivation for me. I am extremely lucky to have a work with him.

I would also like to thank Dr. Somitra Sanadhya for numerous invaluable advice and help during my research. Every discussion with him inspire me to work hard.It is my honour to have worked as a part of the Cryptology Research Group at IIIT-Delhi. I would like to thank every member of CRG@IIITD including Ms. Vartika Srivastva.

Last but not least I would like to thank my family and all my friends for being with me at each step when I need their support. This thesis would never be successful without your support and love.

Contents

List of Figures

Chapter 1

Introduction

Cryptographic primitives, such as hash functions and block ciphers are integral components in the design of practical cryptographic schemes. By using such primitives, cryptosystem can be constructed in secure and efficient manner. Such construction of cryptosystem is easier compared to designing such systems from scratch based on complexity theoretic assumptions. The usual design procedure involves a proposed construction that uses an abstract function/permutation family. The construction is then proven secure by making an appropriate assumption on the function/permutation family. For instance, assuming the function family to be collision resistant or the permutations to be pseudo-random permutations. In practice, these function (resp. permutation) families are instantiated with actual hash functions (resp. block ciphers), in the hope that these constructions will satisfy the required security notion. Depending on the requirements of cryptographic schemes these primitives may need to satisfy a variety of security notions.

The random oracle model was introduced by Bellare and Rogaway as a "paradigm for designing efficient protocols" [2]. This model has proved extremely useful for designing simple, efficient and highly practical solutions for many problems related to designing and security of protocols. From a theoretical perspective, it is clear that a security proof in the random oracle model is only an indication of the security of the system when instantiated with a particular hash function, such as SHA-1 or MD5. In fact, many recent "separation" results [1, 7] illustrated that various cryptographic systems are secure in the random oracle model but completely insecure for any concrete instantiation of the random oracle. In the random oracle model, one proves that the system is at least secure with an "ideal" hash function H (under standard assumptions), instead of making a highly non-standard (and possibly unsubstantiated) assumption that "this system is secure with this specific H" (e.g., H being SHA-1). Such formal proof in the random oracle model is believed to indicate that there are no structural flaws in the design of the system, and thus one can heuristically expect that no such flaws will appear with a particular "well designed" function H. However, it may not guarantee anything about the lack of structural flaws in the design of H itself.

The security proof of a construction as shown in [10] results in a family of collision resistant hash

functions following that construction. The author of [10] claim that a hash function randomly chosen from the hash family along with a random initial string uniformly from the available space can be used as collision resistant hash function.

A hash construction may be of type Hash-then-Truncate(HtT), where first a hash function is applied on input then truncation is applied on output of hash function before producing final output. For example SHA-224 is HtT type construction where internally SHA-256 is used with different IV and output of SHA-256 is truncated to 224 bits. Another example is Chop-Hash construction [9] such as chop-MD.

The HtT construction depends upon a Hash function H and a truncation function T. The truncation function T can be define before defining the H or after it. When T is defined before H, T and H show independence from each other. On the other hand T is defined after H, T may be dependent on H because the definition of T might take care of H properties. This possibility of dependency can raise an question on the randomness of H. This also raises the concern that if truncation depends on H then H may have security flaws or trapdoors; and truncation might have been performed to hide those flaws and trapdoors.

Still, if HtT construction is trapdoor free, the problem of instantiation remains with it. So along with a good design paradigm we also need a secure instantiation methods to have clear security terms. We will explore this need and provide a bridge between theory and practice.

Our Contribution We discuss the design paradigm and security of HtT construction in two different scenarios. In first scenario, hash function H is defined prior to defining the truncation function T and in second scenario T is defined prior to defining H. We prove that first scenario is indifferentiable from random oracle while in second scenario we show attacks by showing an efficient adversary. Next we provide procedure to securely instantiate an HtT construction. In this way, we can also securely instantiate a block cipher based construction.

1.1 Motivation

In SHA-1 the initial chaining vector used is IV= 67452301 EFCDAB89 98BADCFE 10325476 C3D2E1F0. Apparently NIST has taken this IV from MD5. However, how this IV is chosen for MD5 is not well documented. Cryptographers believe that NIST considered some universe of hash functions $\{H_{IV} : IV \in \mathcal{IV}\}$ while choosing the hash function and randomly selected this one design SHA-1, from it. But, NIST never indicated that they did any such thing; all we know is that they selected this one hash function. Theory advocates random sampling to be a crucial element to a valid definition, whereas no random sampling apparently took place in this case.

Similarly, in past too, trust concerns were raised on the definition of S-boxes in DES. Later when differential attack [5] was found on DES, it was realised that the DES designers knew of these attacks already [19].

Attacks like partial collision, near collision, near collision to full collision [11–14] force one to consider evaluation of designing methods, e.g, order of hash function and truncation function definitions in case of HtT construction. All these issues raise doubts on trustworthiness of the designer. Is there some possibility of trapdoor possibility in design?

John Black in [6] showed that instantiated form of a block cipher no longer remains secure because there is always a distinguisher that exists for instantiated block cipher; a fact which is also supported by Phillip Rogaway [18].

To reduce the trust gap between user and designer, we analyse design paradigm taking the order of defining truncation function and hash function as the base. Then we provide a way for secure instantiation of the block cipher along with HtT by utilizing design principle for hash function [10].

Chapter 2

Definitions and Construction

2.1 Indifferentiability

The indifferentiability framework has been introduced by Maurer et al. [15] as a generic tool to study the security of cryptosystems, and its application to the field of hash functions has been first proposed by Coron et al. [9]. Let F be the function that based on an ideal primitive f and R be a random oracle, and S be a simulator with access to R. Then, we say that F^f is indifferentiable from R if for any distinguisher D, there exists an efficient simulator S with negligible probability as follows:

$$\mathrm{Adv}^{indiff}_{F^f,R^S}(\mathrm{D}) = |Pr[D^{F,f} = 1] - Pr[D^{R,S} = 1]| < \epsilon.$$

2.2 Random Oracle Model

R is said to be a random oracle from a set X to set Y if for each x ∈ X the value of R(x) is chosen randomly from Y. More precisely, Pr[R(x)=y|$R(x_1) = y1, R(x_2) = y_2, ...R(x_q) = y_q$] = $\frac{1}{M}$, where x $\notin \{x_1, x_2, \ldots x_q\}$, $y, y_1, \ldots, y_q \in Y$, $|Y| = M$ and q is the total number of queries. If R accepts the variable length input it is consider as VIL Random Oracle

2.3 View

View v is a tuple which consists of pairs (X, Y), where X is of arbitrary length and Y can be of some fixed lengths l or n. More precisely, we can describe view as follows

$$v = ((X_1, Y_1), \ldots, (X_i, Y_i)), \text{ for } j \leq i,\ X_j \in \{0,1\}^* \text{ and } Y_j \in \{0,1\}^l \text{ or } \{0,1\}^n,$$

and V is the set of all views. We consider a distinguisher D which has access of two oracles $\mathcal{O}_1$ and $\mathcal{O}_2$. We assume all quires are distinct and D makes at most q_i queries to oracle $\mathcal{O}_i$. Suppose D makes M_i as $\mathcal{O}_1$-queries and obtains responses $h_i \in \{0,1\}^l$, 1≤i≤ q_1. Similarly, D makes m_i

as $\mathcal{O}_2$-query and obtains responses $z_i \in \{0,1\}^n$, $1 \leq \mathrm{i} \leq q_2$. The obtained after interacting with $\mathcal{O}_1$ and $\mathcal{O}_2$ can be denoted by $v_{\mathcal{O}_1,\mathcal{O}_2}$. We can also denote i-th query-response pair by (X_i, Y_i), where $(X_i, Y_i) = (M_j, h_j)\ for\ a\ 1 \leq j \leq q_1\ OR\ (m_j, z_j)\ for\ a\ 1 \leq j \leq q_2$. So we can define the first i query-response pairs of the tuple v by $v_i = ((X_1, Y_1), \ldots, (X_i, Y_i))$, where v_0 is the empty string. Depending on the view v, the distinguisher D finally outputs 1 or 0.We define $\alpha(v) = Pr[(D(v_{i-1}) = X_i,\ for\ all\ 1 \leq i \leq q_1 + q_2)\&(D(v) = 1)]$. Next we define $\beta_{\mathcal{O}_1,\mathcal{O}_2}(v)$ as the probability that

$$\beta_{\mathcal{O}_1,\mathcal{O}_2}(v) = Pr[\mathcal{O}_1(M_i) = h_i \wedge \mathcal{O}_2(m_j) = z_j,\ for\ 1 \leq i \leq q_1\ and\ 1 \leq j \leq q_2] \tag{2.1}$$

Then Pr[$D^{\mathcal{O}_1,\mathcal{O}_2}$=1] can be also described as follows,

$$Pr[D^{\mathcal{O}_1,\mathcal{O}_2} = 1] = \sum_{v \in V} \alpha(v) \cdot \beta_{\mathcal{O}_1,\mathcal{O}_2}(v) = \sum_{v \in V} \alpha(v) \cdot Pr[v_{\mathcal{O}_1,\mathcal{O}_2} = v]. \tag{2.2}$$

2.4 Truncation function

For any positive integer $a\ and\ b$ such that $a \in \{0,1\}^b$, we represent i^{th} index bit of a as $a[i]$ where $1 \leq i \leq b$.

Truncation function. Let $\tilde{I} = \{1, 2, \ldots, l\}$. A truncation function $T^I : \{0,1\}^l \to \{0,1\}^n$, where I $\subset \tilde{I}$ with $|I| = l - n$, 1≤n≤ l, is defined in Algorithm 1:

Algorithm 1: Truncation function: $T^I(z) = h$

```
1 Initialise: h=empty-string, k=1;
2 for i = 1 → l do
3 |   ifi ∉ I h[k] = z[i];
4 |   k = k + 1
5 return h;
```

Continuous truncation function: We say a truncation function $T^I : \{0,1\}^l \to \{0,1\}^n$ is *continuous* if there exists k, where $1 \leq k \leq (l-n+1)$, such that $I = \tilde{I} - \{k, k+1, \ldots, k+n-1\}$.

2.5 Hash-then-Truncate Construction

Hash-then-Truncate: In the Hash-then-Truncate(HtT) construction as shown in Figure 2.1, $\tilde{H} : \{0,1\}^* \to \{0,1\}^n$ is a function representing HtT construction that accepts a message $M \in \{0,1\}^*$ and gives output $h \in \{0,1\}^n$. $\tilde{H}$ based on $H : \{0,1\}^* \to \{0,1\}^l$ and $T^I : \{0,1\}^l \to \{0,1\}^n$. H accepts the input M from $\tilde{H}$ and gives output $z \in \{0,1\}^l$. Truncation function T^I using specified Index set I takes input z∈ $\{0,1\}^l$ gives output h∈ $\{0,1\}^n$. In short $\tilde{H}(M) = T(H(M))$.

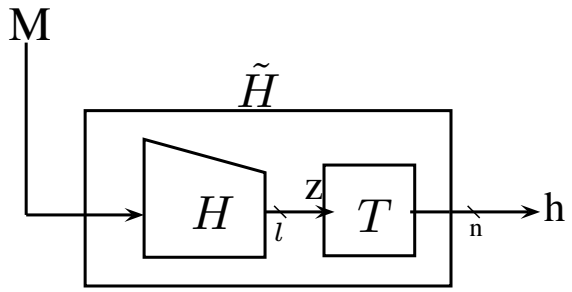

Figure 2.1: Hash-then-Truncate (HtT) construction based on H and T

HtT: $\tilde{H}(M) = h$

00	z=$H(M)$
01	$h = T(z)$
02	return h

Chapter 3

Indifferentiability Proof of HtT Construction

In this chapter we discuss about the indifferentiability of HtT construction. Security of HtT construction depends upon H *and* T. We use the indifferentiability framework of Maurer along with the concepts of view and interpolation probabilities to provide indifferentiability. We follow the approach of indifferentiability proof as explained in [3,8,16] in combine and simple manner. We show that HtT construction is indifferentiable from random oracle when truncation is given.

Theorem 1. *Let T^I be any truncation function as described in Chapter 2.4. After T^I is defined, then independent random oracles H:$\{0,1\}^* \to \{0,1\}^l$ and R:$\{0,1\}^* \to \{0,1\}^n$ are defined. Let $\tilde{H} : \{0,1\}^* \to \{0,1\}^n$ be the function that based on H and T^I, and S is a simulator as described in Figure 3.1.For any distinguisher D with maximum queries q=$q_1 + q_2$ following equation holds for advantage of distinguisher as follows:*

$$Adv^{indiff}_{\tilde{H}^H,R^S}(D) = |Pr[D^{\tilde{H},H} = 1] - Pr[D^{R,S} = 1]| = 0 \tag{3.1}$$

where simulator S has oracle access to R (but does not see the queries of the distinguisher D to R). The distinguisher D makes at most q_1 queries to $\tilde{H}$ or R and at most q_2 queries to H or S

Proof: In Figure 3.1, we have described the $(\tilde{H}, H)$ and R,*S*.

1. *Complexity of Simulator S*: Here complexity is defined by $(O(q), O(q))$ for any distinguisher D with the query-memory-complexity. As shown in Fig. 3.1, the simulator make a query to the random oracle R only when the *S*-query is requested. So, the maximum number of queries of any adversary D is q, so that of the simulator is also $O(q)$. In case of memory, simulator is maintaining a set X, which will have maximum number of elements $O(q)$.

2. Using same notation defined in Chapter 2.3.
   From Equation 2.2 we can write
   $Pr[D^{\tilde{H},H} = 1] = \sum_{v \in V} \alpha(v) \cdot Pr[v_{\tilde{H},H} = v]$ and $Pr[D^{R,S} = 1] = \sum_{v \in V} \alpha(v) \cdot Pr[v_{R,S} = v]$

Algorithm $(\tilde{H}, H)$:	**Algorithm R,*S***:
	Initialise a set X=$\emptyset$
10 On $\tilde{H} - QueryM$	100 **R**- *Query M*
11 z=H(M);	110 h=R(M);
12 h=T(z);	200 ***S***- *Query M*
13 return h.	210 If M is repeated query then return z form X where {(M,z)}$\in$X.
20 On H-*Query M*	220 h=R(M);
21 z=H(M);	230 $p \xleftarrow{\$} \{0,1\}^t$ for a given Truncation function T^I and $t = l - n$
22 return z;	260 k=j=0;
	270 for i=0 to l.
	271 if i$\in$I, then $z[i] = p[k]$ and k++;
	272 else $z[i] = h[j]$ and j++;
	280 X=X$\bigcup(M, z)$
	290 return z.

Figure 3.1: $(\tilde{H}, H)$ and (R,S)

Therefore,

$$\begin{aligned} Adv^{indiff}_{\tilde{H}^H,R^S}(D) &= |Pr[D^{\tilde{H},H} = 1] - Pr[D^{R,S} = 1]| \\ &= |\sum_{v\in V} \alpha(v) \cdot Pr[v_{\tilde{H},H} = v] - \sum_{v\in V} \alpha(v) \cdot Pr[v_{R,S} = v]| \\ &= \sum_{v\in V} \alpha(v)(|Pr[v_{\tilde{H},H} = v] - Pr[v_{R,S} = v]|) \\ &= \sum_{v\in V} \alpha(v)(0) \qquad \textbf{From Claim 1} \\ &= 0 \end{aligned}$$

Claim 1: For any view $v \in V$, where V is the set of all possible as view described in Chapter 2.3

$$Pr[v_{\tilde{H},H} = v] = Pr[v_{R,S} = v], \tag{3.2}$$

Proof of Claim 1: We have assumed that there is no repetition of query to oracle $\mathcal{O}_1$ and $\mathcal{O}_2$ respectively, but a same query can be queried to both oracle $\mathcal{O}_1$ and $\mathcal{O}_2$. From Chapter 2.3, a typical view of $D^{\mathcal{O}_1,\mathcal{O}_2}$ is a tuple $v_{\mathcal{O}_1,\mathcal{O}_2} = ((X_1, Y_1), \ldots, (X_{q_1+q_2}, Y_{q_1+q_2}))$, where $(X_i, Y_i) = (M_k, h_k)$ *or* $(m_{k'}, z_{k'}), 1 \le k \le q_1$ *and* $1 \le k' \le q_2$, where $\mathcal{O}_1(M_k) = h_k \in \{0,1\}^n$, $\mathcal{O}_2(m_{k'}) = z_{k'} \in \{0,1\}^l$, $l > n$, q_1 and q_2 are the number of queries to $\mathcal{O}_1$ and $\mathcal{O}_2$ respectively.

Firstly we define two sets of views- *possible views* V_{pos} and *impossible views* V_{imp} such that $V = V_{pos} \bigcup V_{imp}$ and $V_{pos} \bigcap V_{imp} = \emptyset$.

- We say a view v is impossible view if $\exists((M_k, h_k), (m_{k'}, z_{k'}))$ in v s.t. $(M_i = m_j)$ and $\mathrm{T}(z_j) \neq h_i$.
- We say a view v is possible if $v \notin V_{imp}$.

(a) q' are the number of queries common to $\mathcal{O}_1$ and $\mathcal{O}_2$ with $M_k = m'_k$ more precisely we can say $q' = |\{(i,j) : X_i = X_j \; for \; 1 \le i < j \le q_1 + q_2\}|$.

(b) $(q_1 - q')$ are the number of queries only to $\mathcal{O}_1 : (q_1 - q') = |\{i : (X_i = M_k)\ and\ X_i \neq X_j\ \forall j,\ where\ 1 \leq i \leq q_1 + q_2, 1 \leq j \leq q_1 + q_2, i \neq j\ and\ 0 \leq k \leq q_1\}|$.

(c) $(q_2 - q')$ are the number of queries only to $\mathcal{O}_2 : (q_2 - q') = |\{i : (X_i = m_{k'})\ and\ X_i \neq X_j\ \forall j,\ where\ 1 \leq i \leq q_1 + q_2, 1 \leq j \leq q_1 + q_2, i \neq j\ and\ 0 \leq k' \leq q_2\}|$.

Here $\mathcal{O}_1$ can either be $\tilde{H}$ or R and $\mathcal{O}_2$ can either be H or S. We know from Equation 2.1 and 2.2 $Pr[v_{\mathcal{O}_1,\mathcal{O}_2} = v] = Pr[\mathcal{O}_1(M_k) = h_k \wedge \mathcal{O}_2(m_{k'}) = z_{k'}$, for $1 \leq k \leq q_1$ and $1 \leq k' \leq q_2]$,

i) **In case of** $v \notin V_{imp}$

a) ***For*** $Pr[v_{\tilde{H},H} = v]$: Here $\mathcal{O}_1$ is $\tilde{H}$ and $\mathcal{O}_2$ is H. if $v \notin V_{imp}$ then, for $M_k = m_{k'},\ T(z_{k'}) = h_k$ will always hold.
In case of Random Oracle H, for each new query $\frac{1}{2^l}$ possible values are available, so for $(q_2 - q')$ number of queries to H there will be $Pr[H(m_{k'}) = z_{k'} \forall\ k',\ 0 \leq k' \leq (q_2 - q')] = \frac{1}{2^{(q_2-q')\cdot l}}$.
Similarly for $\tilde{H}$, for every new query $\tilde{H}$ calls H then gives n bit output, so $Pr[\tilde{H}(M_k) = h_k \forall\ k,\ 0 \leq k \leq (q_1 - q')] = \frac{1}{2^{(q_1-q')\cdot n}}$.
For q' number of queries where $M_k = m_{k'}$, $T(z_{k'}) = h_k$ will be satisfied because $v \notin V_{imp}$. Therefore, if $\tilde{H}$ is queried first followed by H, then n bits are chosen randomly in $\tilde{H}$ and only $(l - n)$ bits have to be chosen randomly by H because n bits are already known. Similarly, if H is queried first followed by $\tilde{H}$ then l bits have to be chosen randomly by H as n-bits are already known to $\tilde{H}$. Therefore, $Pr[\tilde{H}(M_k) = h_k \wedge H(m_{k'}) = z_{k'}$ for $M_k = m_{k'}\ and\ T(z_{k'}) = h_k] = \frac{1}{2^{n\cdot q'}} \times \frac{1}{2^{(l-n)\cdot q'}} = \frac{1}{2^{q'\cdot l}}$
Therefore,

$$Pr[v_{\tilde{H},H} = v] = \frac{1}{2^{(q_1-q')\cdot n}} \times \frac{1}{2^{(q_2-q')\cdot l}} \times \frac{1}{2^{q'\cdot l}} \tag{3.3}$$

b) ***For*** $Pr[v_{R,S} = v]$: if $v \notin V_{imp}$ then, for $M_k = m_{k'},\ T(z_{k'}) = h_k$ will always hold.
In case of Random Oracle R , for every new query R gives n bit output, so $Pr[R(M_k) = h_k\ \forall\ k,\ 0 \leq k \leq (q_1 - q')] = \frac{1}{2^{(q_1-q')\cdot n}}$.
For simulator S at line 220 for each new query $\frac{1}{2^l}$ possible values are available, so for $(q_2 - q')$ number of queries to S there will be $Pr[S(m_{k'}) = z_{k'}\ \forall\ k',\ 0 \leq k' \leq (q_2 - q')] = \frac{1}{2^{(q_2-q')\cdot l}}$.
For q' number of queries where $M_k = m_{k'}$, $T(z_{k'}) = h_k$ will be satisfied because $v \notin V_{imp}$. Therefore, if R is queried first followed by S, then n bits are chosen randomly in R and only $(l-n)$ bits have to be chosen randomly by S at line 240 in Figure 3.1 because n bits are already known at line 220 in Figure 3.1. Similarly, if S is queried first followed by R then l bits have to be chosen randomly by S at line 220 in Figure 3.1 as n-bits are already known to R. So $Pr[R(M_k) = h_k \wedge S(m_{k'}) = z_{k'}\ for\ M_k = m_{k'}\ and\ T(z_{k'}) = h_k] = \frac{1}{2^{n\cdot q'}} \times \frac{1}{2^{(l-n)\cdot q'}} = \frac{1}{2^{q'\cdot l}}$. Therefore,

$$Pr[v_{R,S} = v] = \frac{1}{2^{(q_1-q')\cdot n}} \times \frac{1}{2^{(q_2-q')\cdot l}} \times \frac{1}{2^{q'\cdot l}} \tag{3.4}$$

ii) **In case of** $v \in V_{imp}$

a) **For** $Pr[v_{\tilde{H},H} = v]$:if $v \in V_{imp}$, then for common number of queries q', $Pr[\tilde{H}(M_k) = h_k \wedge H(m_{k'}) = z_{k'} \; for \; M_k = m_{k'}$ and $T(z_{k'}) = h_k] = 0$, since for $M_k = m_{k'}$, $T(z_{k'}) \neq h_k$ as per definition of V_{imp}. Therefore,

$$Pr[v_{\tilde{H},H} = v] = 0 \tag{3.5}$$

b) **For** $Pr[v_{R,S} = v]$: if $v \in V_{imp}$, then for common number of quires q' $Pr[R(M_i) = h_i \wedge S(m_j) = z_j \; for \; M_i = m_j$ and $T(z_j) = h_i] = 0$, since for $v \in V_{imp}$ $M_i = m_j$ and $T(z_j) \neq h_i$ as per definition of V_{imp}. Therefore,

$$Pr[v_{R,S} = v] = 0 \tag{3.6}$$

Then, For two cases for any v.

(a) $v \notin V_{imp}$: $\Pr[v_{\tilde{H},H} = v] = Pr[v_{RO,S} = v] = \frac{1}{2^{(q_1 - q') \cdot n}} \times \frac{1}{2^{(q_2 - q') \cdot l}} \times \frac{1}{2^{q' \cdot l}}$ From Equation 3.3 and 3.4

(b) $v \in V_{imp}$: $\Pr[v_{\tilde{H},H} = v] = Pr[v_{RO,S} = v] = 0$ From Equation 3.5 and 3.6

Therefore for any $v \in V$ Claim 1 holds

Chapter 4

Collision Attacks on HtT Construction in the RO model

In Chapter 3 we had discussed the security of HtT construction when T is defined followed by H. Now in this chapter we discuss about the security of HtT construction when H is defined first then T is defined. We will show collision attack on HtT construction when truncation function is allowed to choose after hash function definition.

In this chapter we show that if designer is allowed to choose truncation function T after definition of H, then designer can act as adversary to construct a truncation function such that collision can be found on H and may be used as a trapdoor in $\tilde{H}$.

4.1 In Case when Any Truncation Function is allowed

For any hash function based on random oracle model, there is always a truncation function present which can lead to collision. But for a truncation function it is very negligible to lead collision for all or any one hash function from a set of available hash function.
We can construct an adversary that can make a truncation function using near-collision approach to find partial collision in the H. Here we try to find n-bit collision in l bit output of H and then use $(l-n)$ bit's index to construct truncation function so that $T(H(M)) = T(H(M'))$ can happen and then we can say collision has happened in $\tilde{H}$ or in HtT construction. To explain this we firstly explain some terms briefly that are related to near-collision. This will clear how to find n-bit collision via finding $t = (l-n)$ bit near collision.

A t-bit near collision means that only t-bits are different in hash output of two different messages. A hash function for which an efficient algorithm is known to construct near-collisions, can no longer be considered to be ideal. A practically more relevant consequence is that for several designs, near-collisions for the compression function can be converted to collisions for the hash function.

Some notations that we use are as follows:

l hash function output size, n Truncated output size, t Maximum distance for near-collisions, M_e memory size.

A T^I have index set I that provide $t = l - n$ number of index that have to be truncate. We can define Hamming Weight function HW such that HW(Y)=$|\{Y_i : Y_i = 1, bin_l(Y) = Y_1, Y_2, \ldots, Y_l \ for \ 1 \leq i \leq l\}|$.The probability that a random message M, M' pair results in a t-near collisions is $\mathcal{B}_t(l)/2^l$, where $\mathcal{B}_t(l)$ is Hamming ball [11] of radius t. $\mathcal{B}_t(l) = \#\{x \in \{0,1\}^l : HW(x) \leq t\} = \sum_{i=0}^{t} \binom{l}{i}$ and l is final output of H.

There are different approach to find near collisions that are briefly explained as follows:

Memory-full algorithm. Here M_e has no bound. Compute the hash function on random different inputs a large number of times , and compute the Hamming distance between each pair of outputs. After i evaluations of the hash function, one can test $i(i-1)/2$ pairs, and a pair gives a t-near-collision with probability $\mathcal{B}_t(l)/2^l$. The expected number of computation before finding a near-collision is $i = \sqrt{\pi/2 \cdot 2^l \mathcal{B}_t(l)}$.

Using collisions in a truncated hash function. The hash function is truncated to $s = l - t$ bits, and any collision in the truncated version will give a t-near-collision for the full hash function. More interestingly, if the hash function is truncated to $s = l - 2t - 1$ bits, a s-bit collision will give a t-near-collision of the full hash function with probability 1/2 [12]. This gives a near-collision algorithm with expected complexity $\sqrt{\pi/2 \cdot 2^{l-2t}}$ using a small amount of memory. Generally, one can truncate τ bits, find collisions in a $l - \tau$-bit function, and check the Hamming weight of the τ truncated bits. This will give a t-near-collision with probability $\mathcal{B}_t(\tau)/2^\tau$. The optimal value of τ can be found by evaluating the complexity for all choices of τ. We will treat τ as near-collision truncation parameter. This problem is discussed more formally in [13].

Using covering codes. A more efficient approach is to use covering codes, as proposed by Lamberger et al. [12,13]. The idea is to use a covering code with radius t/2, i.e. a set a codewords $\mathcal{C}$ such that for any point $x \in \{0,1\}^l$, there exists a codeword $c(x) \in \mathcal{C}$ with HW$(x \oplus c(x))\leq$t/2. If $c(H(M)) = c(H(M'))$, then $H(M)$ and $H(M')$ are decoded to the same keyword c; we have HW$(H(M) \oplus H(M')) \leq$HW$(H(M) \oplus c)+$HW$(H(M') \oplus c) \leq t$, which gives a t-near-collision. With a code of dimension k, the attack has a complexity of $\sqrt{\pi/2 \cdot 2^k}$. Finding the optimal τ and building a corresponding code is a hard problem. This problem is discussed by Lamberger et al. in the context of near-collision attacks [12, 13] using a concatenation of Hamming codes. With a given length l, Covering Radius $R = t/2$

$$k = l - R \cdot j - r$$

where, $j := \lfloor log_2(\frac{l}{R} + 1) \rfloor$ and $r := \lfloor \frac{l - R(2^j - 1)}{2^j} \rfloor$.

Theorem 2. *Let l and n be non-zero positive integer such that $l > n$ and t=$(l - n)$.Given Random Oracle H:{0,1}* $\rightarrow${0,1}^{l},We can construct an adversary that can find a truncation*

function T:$\{0,1\}^l \to \{0,1\}^n$ and $M,\ M'$ such that $M \neq M'$ and T(H(M))=T(H(M')) with time complexity
$C_t \leq \left[\left(\sqrt{\frac{\pi}{2}} + 5\sqrt{\frac{2^\tau/\mathcal{B}_{t-2R}(\tau)}{M_e}}\right) \cdot \sqrt{\frac{2^k \cdot 2^\tau}{\mathcal{B}_{t-2R}(\tau)}}\right]$ *and memory complexity* $C_m = C_t \cdot \mathcal{B}_t(\tau)$ *where,* M_e *is available memory,* k *is the code of dimension,* τ *is optimal near-collision truncation parameter,* R *is covering radius.*

Proof : Adversary can follow following steps to make a truncation function.

---

1 Calculate a pair of Message M, M' such that it will give t-near collisions. Adversary can use either Memory Full Algorithm [14], Using Combined approach [14] of Collisions in a truncated hash function [12, 13] and Covering Codes [11, 12]. Using any mentioned method depending upon adversary power and resources, Adversary will have a Message pair M, M';
2 **Initialise:** I=$\emptyset$;
3 Calculate h=H(M), h'=H(M') and $\bar{h} = h \oplus h'$;
4 **for** $i = 0 \to l$ **do**
5 **if** $\bar{h}[i] = 1$ **then**
6 I=I$\bigcup\{i\}$
7 **if** $|I| < t$ **then**
8 **for** $i = 1 \to (t - |I|)$ **do**
9 $i \xleftarrow{\$} \tilde{I} \setminus I$;
10 $I = I \bigcup \{i\}$;
11 Now truncation function is T^I which will follow Index set I.;
12 Calculate h=T($H(M)$) and $\acute{h}$=T($H(M')$), here $h = h'$;

Complexity for Adversary: For adversary main complexity part is to get a message pair M, M' such that HW($H(M)\oplus$H(M'))$\leq$t, Which similar is similar to find t bit near collision. From [14], [11], [12], [13] in which a significant work has been done regarding finding efficient near collisions. We have following methods to find M and M':

1. From [14], if we use Memory Full Algorithm with unbounded available memory M_e then $5\sqrt{\frac{2^\tau/\mathcal{B}_{t-2R}(\tau)}{M_e}}$ become negligible, $\tau = l$,and R=0. The number of hash evaluations needed for any near collision algorithm: we need atleast $i = \sqrt{\pi/2 \cdot 2^l/\mathcal{B}_t(l)}$. After i evaluations, one cant test $i(i-1)/2$ pairs, and a pair gives a t-near collision with probability $\mathcal{B}_t(l)/2^l$. This requires $i \cdot \mathcal{B}_t(l) = \Omega(\sqrt{2^l \cdot \mathcal{B}_t(l)})$ memory acess to a table of size $i = \Omega(\sqrt{2^n/\mathcal{B}_t(l)})$.

2. From [14], in which they have combined the covering-codes and truncation approach technique. To find t-near collisions, truncate hash function to $l - \tau$ and complexity will be $\left(\sqrt{\frac{\pi}{2}} + 5\sqrt{\frac{2^\tau/\mathcal{B}_{t-2R}(\tau)}{M_e}}\right) \cdot \sqrt{\frac{2^k \cdot 2^\tau}{\mathcal{B}_{t-2R}(\tau)}}$, where k is the code of dimensions as explained in [12] which are related to t-near collision.

Example 1: In case of SHA-224, there is 32 bit truncation done on SHA-256. We can find a truncation which enables the attacker to find a collision with approximately 2^{75} complexity in

case of memory full algorithm and near around $2^{83.7}$ complexity when 1MB memory complexity, which is less than the birthday attack complexity.

4.2 In Case when Only Continuous Truncation Function is allowed

In chapter 4.1, we have considered truncation function that truncate any bit position combination from z, now we consider continuous truncation function that will output only continuous n no. of bits from z. Such truncation is more in practical use where after hash computation continues n-bit gives as output.

Depending upon the value of l and n, an adversary can construct a truncation function such that $H(M) \oplus H(M')$ will give continues n-bit collisions. The probability that $H(M) \oplus H(M')$results in n-bit continuous collisions is $i_{ct}/2^l$

$$i_{ct} = \sum_{i=0}^{l-n} d_i \times 2^{l-n-i}. \tag{4.1}$$

where, $d_0 = 1$, $d_i = 2^i - \left[\sum_{j=1}^{i} 2^{\langle \frac{i-j}{n} \rangle} \cdot d_{j-1}\right]$ and $\langle \cdot \rangle$ shows positive integer value.

Theorem 3. *Let l and n be non-zero positive integer such that $l > n$ and $t=(l-n)$. Given random oracle $H:\{0,1\}^* \rightarrow \{0,1\}^l$, we can construct an adversary that can find continuous t bit truncation function $T:\{0,1\}^l \rightarrow \{0,1\}^n$ and $M,\ M'$ such that $M \neq M'$ and $T(H(M))=T(H(M'))$ with time complexity $C_{ct} \geq \sqrt{\pi/2 \cdot 2^l/i_{ct}}$ and memory complexity $(C_{ct} \cdot i_{ct})$.*
where, $i_{ct} = \sum_{i=0}^{l-n} d_i \times 2^{l-n-i}$, $d_0 = 1$ and $d_i = 2^i - \left[\sum_{j=1}^{i} 2^{\langle \frac{i-j}{n} \rangle} \cdot d_{j-1}\right]$.

Proof : Adversary can follow following steps to make a truncation function.

Complexity of Attack: For adversary main complexity part is to get a message pair M, M' such that $\mathrm{HW}(H(M) \oplus \mathrm{H}(M')) \leq (l-n)$ and continuous n-bit collision out of l bit. After s evaluation one can test $s(s-1)/2$ pairs and a pair will give n-bit continues collision with probability $i_{ct}/2^l$. The expected number of computation before finding a n-bit continuous collision is $s = \sqrt{\pi/2 \cdot 2^l/i_{ct}}$. This will give a lower bound on the number of hash evaluations needed for n-bit continuous collision we need atleast $\sqrt{\pi/2 \cdot 2^l/i_{ct}}$ evaluations with non-negligible probability. This approach will require $s.i_{ct}$ memory access to a table of size $\Omega(\sqrt{2^l/i_{ct}})$. So if we assume each hash computation as unit one the time complexity of finding Continuous Truncation function will have lower bound $\sqrt{\pi/2 \cdot 2^l/i_{ct}}$ and memory complexity will be $\Omega(\sqrt{2^l/i_{ct}})$.

Example 2: In case of SHA-224, there is 32 bit truncation done on SHA-256. We can find a truncation which enables the attacker to find a collision with $2^{110.6}$ complexity, which is less than the birthday attack complexity.

```
Calculate a pair of Message M, M' such that it will give n-bit continuous collisions after
H(M), H(M'). Adversary can use Equation 4.1. Adversary will have a Message pair
M, M';
Initialise: I=∅, x=1, c=0;
Calculate h̄ = H(M) ⊕ H(M');
for i = 0 → l do
    if h̄[i] = 0 then
        if c = n then
            x=i-1;
            end for;
        c++;
    else
        if c = n then
            x=i-1;
            end for;
        c=0;
k=x-n+1;
I=Ĩ − {k, k+1, ..., k+n−1};
Now truncation function is T^I which will follow Index set I.;
Calculate h=T(H(M)) and h'=T(H(M')), here h = h'.;
```

Chapter 5

Secure Instantiation of HtT construction

In [6], author showed that a block-cipher based hash function is provably-secure in ideal cipher model but trivially insecure when instantiated by any block-cipher. In fact, many results [1, 7] illustrated various cryptographic systems secure in the random oracle model but completely insecure for any concrete instantiation of the random oracle. They have raised questions on wisdom of security proofs of block ciphers which might get failed once block cipher get instantiate.

We can go back to 90's when Damgard [10] gives the design principle for hash function, in paper they have proved the security of hash family from which a hash function is chosen randomly and used. Similarly, in [18] Rogaway raised a question concerns how SHA-1 is said to be randomly selected from universe of hash function. They have selected only one hash function out of hash family. Rogaway also said that there always exist a adversary that can find collision in a hash function, its just a human that can't actually write the program for it, this statement was supported by Jhon Black in [6] by proving insecurity of a hash function once it get instantiated.

Protocol uses crypto hash functions are often proved secure in RO model [2]. In such a case, when one replace the RO with a concrete hash function(like SHA-1) one would like to preserve the function's domain and range; H:X$\rightarrow$Y for X,Y$\subseteq \{0,1\}^*$. So replacing a RO by a concrete hash function always takes you away from hash-family setting. All security proofs of schemes are done for a hash family and that proof's result is used for a single instantiated hash function assuming the instantiated scheme is secure as long as instantiated hash function behaves "like a random oracle", which is not secure and might even have trapdoor. So what's the solution? How we can achieve proved security? How we can trust the designer that he doesn't put any trapdoors.

A possible solution is as follows; after security proof and before instantiate block cipher there should be a step of choosing a hash function from a hash family for user in between, this will also lead to independence of truncation from hash function which is proved to be secure. This generate a need of "*having a hash family*".

5.1 General Approach

Now we will give some general approach for creating a hash family. Suppose we have a finite index space C. We define a Hash Family $\bar{H} = \{H_c : \{0,1\}^* \to \{0,1\}^l\}_{c \in C}$. Here Hash function H_c uses a value $c \in C$ internally, where c might be an initial value, constant value, key value, chaining value etc.

1. **Changing Initial Value Used**: We can denote the initial value as IV, then we can treat hash family as $\bar{H} = \{H_{IV} : \{0,1\}^* \to \{0,1\}^l\}_{IV \in C}$ names a function $H_{IV}(\cdot) = H(IV, \cdot)$, l is the output length of Hash function H.

   For example in Keccak-1600 [4], $\bar{H} = \{H_c : \{0,1\}^* \to \{0,1\}^l\}_{c \in C}$, where $C = \{c : c \in \{0,1\}^{1600}\}$. Here size of hash family will be 2^{1600}. But Keccak uses only the 0^{1600} value from the C.

   If designer restricts the user to some specific initial value then it can raise the question on randomness and values distribution of hash function and there is chance of having trapdoors in construction. After having choice of hash function from hash family user can freely choose any H and along with public IV. Here each $H \in \bar{H}$ use same instantiation but different starting point, so user can freely start using randomness distribution of instantiated hash function. But still the control of instantiation is with designer because we have not modified the instantiation part. It is very likely that any method would cause collisions in one Keccak instantiation with respect to the actual Initial Value(IV) would work for other IV values too. Intuitively we can say that if there is trapdoor with one single starting point i.e, IV value then there can also be trapdoors on other IV values too. As instantiation part is not changed so we can not justify the secure instantiation. Now we will discuss second approach.

2. **Changing constant value**: A Message M after padding $pad(M) = (m_1 \| m_2 \| \dots \| m_k)$, where $pad(M) \in (\{0,1\}^b)^k$, $m_i \in \{0,1\}^b$ and k is called as number of blocks of M after padding. Given a function $f : \{0,1\}^{l+b} \to \{0,1\}^l$ we define

   $$H^f(m_1, m_2, \dots, m_k) = f(f(\dots f(f(IV, m_1), m2), \dots), m_k)$$

   where H is Hash function with underlying compression function f, having an initial value IV. We can define the family of function $\bar{f} = \{f_c : c \in C\}$ where each f uses some constant $c \in C$ internally. Further, if function f uses a randomly chosen constant $c \in C$ we will denote that as f_c, otherwise for a function using fixed constant c we will denote it as simple f. We will now define a Hash Family $\bar{H} = \{H^{f,f_c} : \{0,1\}^* \to \{0,1\}^l\}_{c \in C}$, where Hash function H is based on combination of f and f_c.

   Once the instantiated scheme is proposed, we use a random value of $c \xleftarrow{\$} \{0,1\}^l$ that will be used in f_c to randomise the instantiation of f and H, simultaneously decreasing the possibility of presence of trapdoors in instantiation. Different f can have different c value depending upon provided instantiation.

For example in MD5 [17] for single block message processing, there will be total 64 rounds of f, depending upon round, different fixed constant c is used in different rounds of f, like for round 0 to 3, $f_{[0,\ldots,3]}$ uses $[0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee]$. It is already assumed that these constants are fixed but chosen randomly. Now according to our approach in-spite of using all fixed constant proposed by instantiated scheme, we randomly choose constant c in some f and denote them as f_c. Here each constant size is 32-bit, therefore hash family size will be 2^{32} if we use only one f_c, which is still not enough. We can increase the family size by including more then one f_c, then size of hash family will be $2^{32*(\#f_c)}$.(# number of)

Now we will give a definition of adversary that will find the truncated collision of hash family and truncation family, as follows.

Definition 1. *Let $\bar{T} = \{T : \{0,1\}^l \to \{0,1\}^n\}$ be a truncation function family for positive integer $l > n$. We have a Message space $\mathcal{M} = \{M \in \{0,1\}^*\}$. We assume that $\mathcal{M}$ contains some string of length greater then l and that $X \in \mathcal{X}$ implies every string of length $|M|$ is in $\mathcal{M}$. Let $\bar{H}$ is the family of hash function such that $\bar{H} = \{H^{f,f_c} : \{0,1\}^* \to \{0,1\}^l\}_{c\in C}$, where H is a hash function with underlying compression function $f, f_c \in \bar{f} = \{f_c : \{0,1\}^{b+l} \to \{0,1\}^l\}_{c\in C}$, C is the finite index set, b is the message block size taken by f and l is the output length of hash function H. After choosing an arbitrary truncation function $T \in \bar{T}$ by adversary $\mathcal{A}$ then advantage of Adversary $\mathcal{A}$ with respect to truncated collision resistant for a randomly chosen hash function $H \in \bar{H}$ is given by*

$$Adv^{t-coll}_{\bar{H},\bar{T}}(\mathcal{A}) = Pr[T \xleftarrow{arb.} \mathcal{A}; H \xleftarrow{\$} \bar{H}; M, M' \leftarrow \mathcal{A} : T(H(M)) = T(H(M')) \wedge M \neq M'] \quad (5.1)$$

In further chapter we will provide a construction as an example of second approach.

5.2 Construction

Here we will provide a Hash-then-Truncate construction as shown in Figure 5.1 as an example of second approach described above, which can be securely instantiated.

First we define the mechanism. Let $f : \{0,1\}^{b+l} \to \{0,1\}^l$ be an hash function, called a compression function, and define from it the Hash function $H : \{0,1\}^* \to \{0,1\}^l$ as follows. On input $M \in \{0,1\}^*$, H partitions message M using padding function $\mathbf{pad}(M)$=$M\|0^p\|[|M|]_b$ into b-bit strings $m_1, \ldots, m_k$ where $p \geq 0$ is the least non-negative number such that $|M| + p$ is a multiple of b. We have a finite index set $C = \{c : c \in \{0,1\}^l\}$. For some initial value $IV \in C$ we can more formally say that

$$H^{f,f_c}(m_1, m_2, \ldots, m_k) = f_c(m_k, f(\ldots f(m_2, f(m_1, IV))))$$

Here construction will use f_c function only for processing the last message block m_k, which means only for last message block m_k function f will use a randomly chosen constant c internally while

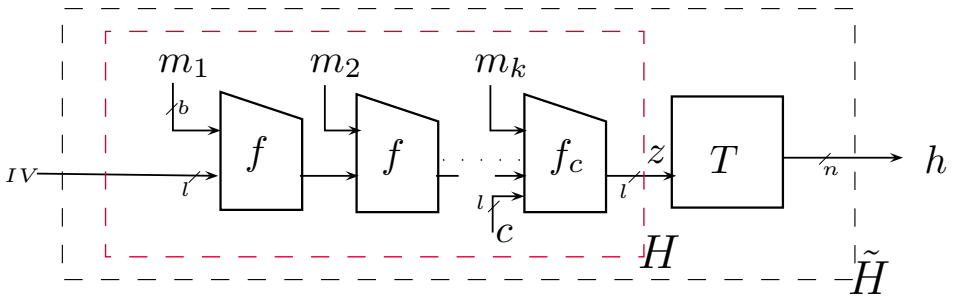

Figure 5.1: Modified Hash-then-Truncate (HtT) construction-Domain Extension

for rest function f will use some fixed constant. So here we have an Hash Family $\bar{H} = \{H^{f,f_c} : \{0,1\}^* \to \{0,1\}^l\}_{c \in C}$, of size $|C|$. Then, letting $z_0 = IV$, define $z_{i-1} = f(m_{i-1}, z_{i-2})$ for each $i \in [2..k]$ and then $z_k = f_c(m_k, z_{k-1})$. So $H(M)$ returns z_k. Now we have a truncation function as per defined in chapter 2.4 . Let $\tilde{I} = \{1, 2, \ldots, l\}$. A truncation function $T^I : \{0,1\}^l \to \{0,1\}^n$, where I $\subset \tilde{I}$ with $|I| = l - n$, $1 \leq n \leq l$. After choosing an arbitrary Truncation function T^I first and then choosing a random Hash function $H \xleftarrow{\$} \bar{H}$, we can define a Hash-then-Truncate construction $\tilde{H} : \{0,1\}^* \to \{0,1\}^n$, based on Hash function H and a truncation function T^I. More formally $\tilde{H}(M) = T(H(M))$.

Algorithm 2: Modified Hash-then-Truncate HtT construction

Initialise: $z_0 = IV$; $T \xleftarrow{\$} \bar{T}$; $f_c \xleftarrow{\$} \bar{f}$
for *any* M, $T(H^{f,f_c}(M)) = h$ *is calculated as follows:* **do**
    pad(M)$=m_1, m_2, m_3, \ldots, m_k$, where $|m_i| = b$, $i \in [1, \ldots, k]$
    for $i = 1 \to (k-1)$ **do**
        $z_i = f(m_i, z_{i-1})$
    $h = T(f_c(m_k, z_{k-1}))$
    Return h

Now we give an experiment where any adversary $\mathcal{A}$ finds the collision on given construction following Definition 1. In experiment adversary chooses truncation function T arbitrary from $\bar{T}$. Then adversary chooses a hash function $H \xleftarrow{\$} \bar{H}$, where H is based on compression function f and f_c. $\mathcal{A}$ tries to return a colliding message pair M, M' such that $T(H^{f,f_c}(M)) = T(H^{f,f_c}(M'))$.

We have Event 1 and Event 2 as follows:

Definition 2. *Event 1 is the event that there exists a collision pair of $f(\cdot,\cdot)$ from a message pair (M, M') generated by adversary $\mathcal{A}$*

Definition 3. *Event 2 is the event that there exists a collision pair of $T(f_c(\cdot,\cdot))$ from a message pair (M, M') generated by adversary $\mathcal{A}$*

Following Construction as per Algorithm 2 and Definition 1, if adversary $\mathcal{A}$ succeeds in getting

two message M, M' as colliding pair then collision happen either due to $f(\cdot)$ or $T(f_c(\cdot))$. See after message length padding

$pad(M) = m_1, m_2, \ldots, m_k$

$pad(M') = m'_1, m'_2, \ldots, m'_{k'}$

1. If M, M' is collision of $T(H^{f,f_c}(\cdot))$ and $|M| \neq |M'|$, then always $m_k \neq m'_{k'}$ due to message length padding and $(m_k, z_k), (m'_{k'}, z'_{k'})$ will result in collision of $T(f_c(\cdot,\cdot))$

2. If M, M' is collision of $T(H^{f,f_c}(\cdot))$ and $|M| = |M'|$, then

   $$\begin{aligned} T(f_c(m_k, z_{k-1})) &= \tilde{H}(M) \\ &= \tilde{H}(M') \\ &= T(f_c(m'_{k'}, z'_{k'-1})) \end{aligned}$$

   Either here $(m_k \| z_{k-1}) \neq (m'_{k'} \| z'_{k'-1})$ means we have found collision on $T(f_c(\cdot))$
   Or $(m_k \| z_{k-1}) = (m'_{k'} \| z'_{k'-1})$ then we go more backward

   $$\begin{aligned} f(m'_{k'-1}, z'_{k'-2}) &= z'_{k'-1} \\ &= z_{k-1} \\ &= f(m_{k-1}, z_{k-2}) \end{aligned}$$

   Either here $(m_{k-1} \| z_{k-2}) \neq (m'_{k'-1} \| z'_{k'-2})$ means collision of f
   Or $(m_{k-1} \| z_{k-2}) = (m'_{k'-1} \| z'_{k'-2})$ then we go more backward with $z_{k-2} = z'_{k'-2}$ until $(m_{i-1} \| z_{i-2}) \neq (m'_{j'-1} \| z'_{j'-2})$ where $i \in [k-1, \ldots, 1]$ and $j \in [k'-1, \ldots, 1]$ which will show definitely collision on f due to equality of truncated hash output.

Precisely we can say, since $\mathcal{A}$ finds $\tilde{H}(M) = \tilde{H}(M')$, where $\tilde{H}$ is based on T and H^{f,f_c}, therefore collision on $\tilde{H}$ is due to collision on H or $T(H(\cdot))$.

Further H is based on f and f_c. So collision on H either due to internal collision on f or f_c. If collision is not due to internal collision on f, f_c, then $(l-n)$-bit near collision on f_c output z will lead to collision after truncation function T. If we combine overall then collision on $\tilde{H}$ either due to collision on $f(\cdot)$ or $T(f_c(\cdot))$. Therefore truncated collision will be only due to f or $T(f_c(\cdot))$. Therefore we can write Equation 5.1 as follows

$$\begin{aligned} Adv^{t-coll}_{\tilde{H},\tilde{T}}(\mathcal{A}) &= Pr[T \xleftarrow{\text{arbiratory}} \mathcal{A}; H \xleftarrow{\$} \bar{H}; M, M' \leftarrow \mathcal{A} : T(H(M)) = T(H(M')) \wedge M \neq M'] \\ &= Pr[Event\ 1\ or\ Event\ 2] \\ &\leq Pr[Event\ 1] + Pr[Event\ 2] \end{aligned}$$

$$Adv^{t-coll}_{\tilde{H},\tilde{T}}(\mathcal{A}) \leq Pr[Event\ 1] + Pr[Event\ 2] \tag{5.2}$$

Definition 4. *Let $\mathcal{B}$ be an adversary for attacking $f : \{0,1\}^{b+l} \to \{0,1\}^{l}$,meaning an algorithm that outputs a pair of strings $(m,z),(m',z')$. We let the advantage of $\mathcal{B}$ in finding collision in f as*

$Adv_f^{col}(\mathcal{B}) = Pr[(m,z),(m',z') \leftarrow \mathcal{B} : [f(m,z) = f(m',z')] \wedge [(m,z) \neq (m',z')]]$

that measure the chance that $\mathcal{B}$ finds a collision.

We will represent *col* as collision on function and *coll* as collision on function family [18].

Claim 2: When adversary $\mathcal{A}$ succeeds in generating a colliding pair message M, M', then a collision finding adversary $\mathcal{B}_{\mathcal{A}}$ can exist on f, explicitly given in proof where $Pr[Event\ 1] = Adv_f^{col}(\mathcal{B}_{\mathcal{A}})$.

Proof of **Claim 2**: We have an adversary $\mathcal{A}$ output a colliding Message pair M, M', for $\tilde{H}(\cdot) = T(H^{f,f_c}(\cdot))$ given T and $H^{f,f_c} \in \bar{H}$. Now we can construct a collision finding adversary $\mathcal{B}_{\mathcal{A}}$ on f that will return $((m,z),(m',z'))$ as follows Algorithm 3.

Algorithm 3: Adversary $\mathcal{B}_{\mathcal{A}}$

```
z_0 = z'_0 = IV, T ← A
B_A chooses c randomly from set C and return c to A
H^{f,f_c} is given to A
M, M' ← A
if |M'| = |M| then
    pad(M) = m_1, m_2, ..., m_k and pad(M') = m'_1, m'_2, ..., m'_{k'}
    if m_k = m'_{k'} then
        for i = 1 → (k − 1) do
            z_i = f(m_i, z_{i−1})
            z'_i = f(m'_i, z'_{i−1})
        for i = (k − 1) → 1 do
            if (m'_i ≠ m_i) ∧ (z'_i = z_i) then
                Return :((m'_i, z'_{i−1}), (m_i, z_{i−1}))
        Fails;
    else
        Fails;
else
    Fails;
```

In Algorithm, adversary $\mathcal{A}$ chooses a truncation function $T \in \bar{T}$ and get a $c \in C$ from $\mathcal{B}$. Using c, f and T, $\mathcal{A}$ get $H^{f,f_c} \in \bar{H}$ and find a message colliding pair M, M' on $\tilde{H} = T(H^{f,f_c}(\cdot))$. Now $\mathcal{B}$ uses M, M' to find the existence of collision on f. After padding M and M' have k and k' number of $b-$bit blocks respectively assuming $k' \leq k$.

If $|M| \neq |M|$ and $m_k \neq m'_{k'}$ then collision will be Event 2 and it is not the intrest case of $\mathcal{B}_{\mathcal{A}}$. So if $|M| = |M|$ and $m_k = m'_{k'}$ then only collision will be Event 1. As $k' = k$, we start

from $i = (k-1)$ to check collision is happening on f or not. If for any $i \in [(k-1), \ldots, 1]$ $(m'_i \neq m_i) \wedge (z'_i = z_i)$ happens will result in collision on f at $((m'_i, z'_{i-1}), (m_i, z_{i-1}))$.

So advantage of adversary $\mathcal{B}$ have that will give probability of Event 1 happening

$Pr[Event\ 1] = Adv_f^{col}(\mathcal{B}_\mathcal{A})$
$Adv_f^{col}(\mathcal{B}_\mathcal{A}) = Pr[H \xleftarrow{\$} \bar{H}; (m,z), (m',z') \leftarrow \mathcal{B}_\mathcal{A} : [f(m,z) = f(m',z')] \wedge [(m,z) \neq (m',z')]]$

Claim 3:When adversary $\mathcal{A}$ succeed in generating a colliding pair message, then a collision finding adversary $\mathcal{C}_\mathcal{A}$ can exist on $T(f_c(\cdot))$, explicitly given in proof where $Pr[Event\ 2] = Adv_{\bar{f},\bar{T})}^{t-coll}(\mathcal{C}_\mathcal{A})$

Proof of **Claim 3**: We have an adversary $\mathcal{A}$ that will output a Message pair M, M' as a truncated-colliding pair, for a given $T(H^{f,f_c}(\cdot))$. Now we can construct collision finding an adversary $\mathcal{C}_\mathcal{A}$ on f_c that will return $((m,z),(m',z'))$ as follows Algorithm 4.

Algorithm 4: Adversary $\mathcal{C}_\mathcal{A}$

```
z_0 = z'_0 = IV, T ← A and return T to C_A
c is given to C_A, where c ←$ C and return c to A
M, M' ← A
if |M'| > M then
    Swap(M, M')
pad(M) = m_1, m_2, ..., m_k and pad(M') = m'_1, m'_2, ..., m'_{k'}
for i = 1 → (k − 1) do
    z_i = f(m_i, z_{i−1})
for j = 1 → k' − 1 do
    z'_j = f(m'_j, z'_{i−1})
if m_k ≠ m'_{k'} ∨ z_{k−1} ≠ z'_{k'−1} then
    if T(f_c(m_k, z_{k−1}))) = T(f_c(m'_{k'}, z'_{k'−1})) then
        Return ((m_k, z_{k−1}), (m'_{k'}, z'_{k'−1}))
    else
        Fails;
else
    Fails;
```

In algorithm, if $m_k \neq m'_{k'}$ then it definitely means collision on $T(f_c(\cdot,\cdot))$. If $m_k = m'_{k'} \wedge z_{k-1} \neq z'_{k'-1}$ then $T(f_c(m_k, z_{k-1})) = T(f_c(m'_{k'}, z'_{k'-1}))$ must happen in order to satisfy collision pair M, M' of $T(H(\cdot))$ given by adversary $\mathcal{A}$. As explained earlier in construction $c \xleftarrow{\$} C$ will result into $f_c \xleftarrow{\$} \bar{f}$. Therefore advantage of adversary $\mathcal{C}$ will give the probability of Event 2 as follows.
$Pr[Event\ 2] = Adv_{\bar{f},\bar{T}}^{t-coll}(\mathcal{C}_\mathcal{A})$
$= Pr[T \xleftarrow{\text{arb.}} \mathcal{A}; f_c \xleftarrow{\$} \bar{f}; (m,z),(m',z') \leftarrow \mathcal{C}_\mathcal{A} : [T(f_c(m,z)) = T(f_c(m',z'))] \wedge [(m,z) \neq (m',z')]]$

We now show that if $f, \bar{f}$ is collision resistant then $\bar{H}$ is truncated-collision resistant family for

any given $T \in \bar{T}$.

Theorem 4. *There exists algorithm $\mathcal{B}$ and $\mathcal{C}$, explicitly given in the proof of this theorem. such that for a Truncation family $\bar{T} = \{T : \{0,1\}^l \to \{0,1\}^n\}$ and Hash Family $\bar{H} = \{H^{f,f_c} : \{0,1\}^* \to \{0,1\}^l\}_{c\in C}$, given a Index set $C = \{c : c \in \{0,1\}^l\}$, a compression family $\bar{f} = \{f_c : \{0,1\}^{b+l} \to \{0,1\}^l\}_{c\in C}$, and any adversary $\mathcal{A}$, adversaries $\mathcal{B}_\mathcal{A}$ and $\mathcal{C}_\mathcal{A}$ satisfy*

$$Adv^{t-coll}_{\bar{H},\bar{T}}(\mathcal{A}) \leq Adv^{col}_{f}(\mathcal{B}_\mathcal{A}) + Adv^{t-coll}_{\bar{f},\bar{T}}(\mathcal{C}_\mathcal{A}) \tag{5.3}$$

Proof. From Definition 1,

$$Adv^{t-coll}_{\bar{H},\bar{T}}(\mathcal{A}) = Pr[T \xleftarrow{\text{arb.}} \mathcal{A}; H \xleftarrow{\$} \bar{H}; M, M' \leftarrow \mathcal{A} : T(H(M)) = T(H(M')) \wedge M \neq M'] \tag{5.4}$$

If Adversary $\mathcal{A}$ succeed then collision can be happen at either f or truncated collision on f_c. So we can use Equation 5.2

$Adv^{t-coll}_{\bar{H},\bar{T}}(\mathcal{A}) \leq Pr[Event\ 1] + Pr[Event\ 2]$

Using **Claim 2** and **Claim 3** we can say that

$$Adv^{t-coll}_{\bar{H},\bar{T}}(\mathcal{A}) \leq Adv^{col}_{f}(\mathcal{B}_\mathcal{A}) + Adv^{t-coll}_{\bar{f},\bar{T}}(\mathcal{C}_\mathcal{A})$$

We have shown that whenever $\mathcal{A}$ outputs a collision of $\bar{H}$, either adversary $\mathcal{B}_\mathcal{A}$ or $\mathcal{C}_\mathcal{A}$ outputs collision on f and truncated collision on f_c respectively. □

Finally in this chapter we showed that we need a family of hash functions to guarantee the security against the truncated collision-finding attackers, where advantage of adversary should be negligible over truncation family and hash family. Without hash family, we cannot expect any security proof for any truncation because of instantiation problem. So we gave hash domain extension of truncated hash function along with the reduction proof. We can provide a hash family where we can instantiate the hash family by changing constants.

Chapter 6

Conclusion

For practical usage of designs and there security proof, we need a hash family that can be instantiated and then be ready for use by choosing a random hash function from that. For creating a hash family we can use changing constant method. In similar way we can instantiate a Hash-then-Truncation construction along with following correct order of defining hash function and truncation function. All these basic norms can help in proposing trusty design along with robust security.

Bibliography

[1] Bellare, M., Boldyreva, A., and Palacio, A. An uninstantiable random-oracle-model scheme for a hybrid-encryption problem. In *Advances in Cryptology - EUROCRYPT 2004*, C. Cachin and J. Camenisch, Eds., vol. 3027 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2004, pp. 171–188.

[2] Bellare, M., and Rogaway, P. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security* (1993), D. E. Denning, R. Pyle, R. Ganesan, R. S. Sandhu, and V. Ashby, Eds., ACM, pp. 62–73.

[3] Berstein, D. J. A short proof of the unpresdictability of cipher block chaning. `http://cr.yp.to/antifrogery/easycbc-20050109.pdf`.

[4] Bertoni, G., Daemen, J., Peeters, M., and Assche, G. V. On the indifferentiability of the sponge construction. In *EUROCRYPT* (2008), N. P. Smart, Ed., vol. 4965 of *Lecture Notes in Computer Science*, Springer, pp. 181–197.

[5] Biham, E., and Shamir, A. Differential cryptanalysis of des-like cryptosystems. In *CRYPTO* (1990), A. Menezes and S. A. Vanstone, Eds., vol. 537 of *Lecture Notes in Computer Science*, Springer, pp. 2–21.

[6] Black, J. The ideal-cipher model, revisited: An uninstantiable blockcipher-based hash function. In *FSE* (2006), M. J. B. Robshaw, Ed., vol. 4047 of *Lecture Notes in Computer Science*, Springer, pp. 328–340.

[7] Canetti, R., Goldreich, O., and Halevi, S. The random oracle methodology, revisited. *J. ACM 51*, 4 (July 2004), 557–594.

[8] Chang, D., and Nandi, M. Improved indifferentiability security analysis of chopmd hash function. In *FSE* (2008), K. Nyberg, Ed., vol. 5086 of *Lecture Notes in Computer Science*, Springer, pp. 429–443.

[9] Coron, J.-S., Dodis, Y., Malinaud, C., and Puniya, P. Merkle-damgård revisited: How to construct a hash function. In *CRYPTO* (2005), V. Shoup, Ed., vol. 3621 of *Lecture Notes in Computer Science*, Springer, pp. 430–448.

[10] Damgård, I. A design principle for hash functions. In *CRYPTO* (1989), G. Brassard, Ed., vol. 435 of *Lecture Notes in Computer Science*, Springer, pp. 416–427.

[11] Lamberger, M., Mendel, F., Rijmen, V., and Simoens, K. Memoryless near-collisions via coding theory. *Des. Codes Cryptography 62*, 1 (2012), 1–18.

[12] Lamberger, M., and Rijmen, V. Optimal covering codes for finding near-collisions. In *Selected Areas in Cryptography* (2010), A. Biryukov, G. Gong, and D. R. Stinson, Eds., vol. 6544 of *Lecture Notes in Computer Science*, Springer, pp. 187–197.

[13] Lamberger, M., and Teufl, E. Memoryless near-collisions, revisited. *Inf. Process. Lett. 113*, 3 (2013), 60–66.

[14] Leurent, G. Time-memory trade-offs for near-collisions. *IACR Cryptology ePrint Archive 2012* (2012), 731.

[15] Maurer, U. M., Renner, R., and Holenstein, C. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In *TCC* (2004), M. Naor, Ed., vol. 2951 of *Lecture Notes in Computer Science*, Springer, pp. 21–39.

[16] Nandi, M. A simple and unified method of proving indistinguishability. In *INDOCRYPT* (2006), R. Barua and T. Lange, Eds., vol. 4329 of *Lecture Notes in Computer Science*, Springer, pp. 317–334.

[17] Rivest, R. The md5 message-digest algorithm, 1992.

[18] Rogaway, P. Formalizing human ignorance. In *VIETCRYPT* (2006), P. Q. Nguyen, Ed., vol. 4341 of *Lecture Notes in Computer Science*, Springer, pp. 211–228.

[19] Wikipedia. Data encryption standard. `http://en.wikipedia.org/wiki/Data_Encryption_Standard`.