CS 641 : ASSIGNMENT : PART 7 : SCHUTZSAS

A PREPRINT

Srinjay Kumar
170722

Abhyuday Pandey
170039

Kumar Shivam
170354

June 15, 2020

1 Terminology

1.1 Preimage Attack

Preimage attack on a cryptographic hash functions tries to find a message that has a specific hash value. A cryptographic hash function should resist attacks on its preimage

1.2 Second Preimage Attack

Second Preimage attack on a cryptographic hash functions tries to find a message that has same specific hash value as that of the specified input, i.e., given x, we have to to find a second preimage $x \neq x'$ such that $h(x) = h(x')$. If we can find two preimages for a given hash value, we are done. This, because we can find two preimages for $h(x)$, even in the case when one of the preimage coincides with x, we can report the other preimage.

1.3 Collision Attack

A collision attack on a cryptographic hash tries to find two inputs producing the same hash value Find two different messages m_1 and m_2 such that $hash(m_1) = hash(m_2)$.

1.4 Slice notation for array

We follow the slice notation for array in the later part of this report. This means that if a is an array of length n, $a[i : j]$ refers to part of array from index i to index j. Incase, i is missing it means $i = 0$, and if j is missing it means $j = n - 1$. It is important to note that i and j may be negative numbers, which means counting proceeds from backward. This scheme is used in `Python`.

1.5 Modulo indexing

x and y indexing are modulo 5 while z indexing are modulo 8. Explicit mention of modulo at each index is avoided to maintain clarity.

2 Inverting χ

Let us first define what χ is. Let $A[x, y, z]$ where $x \in \mathbb{Z}_5, y \in \mathbb{Z}_5$ and $z \in \mathbb{Z}_8$ be a bit in the cube, the transformation χ is defined by,

$$\chi(A[x, y, z]) = A[x, y, z] \oplus (1 \oplus A[x + 1, y, z]).A[x + 2, y, z]$$

Consider keeping (y, z) fixed, i.e. looking at the row. Let a_0, a_1, a_2, a_3, a_4 be the input row and b_0, b_1, b_2, b_3, b_4 be the output row. Assuming the indices are from $\mathbb{Z}_5$, we have

$$b_i = a_i \oplus (1 \oplus a_{i+1}).a_{i+2}$$

We will try to invert the same,

$$\begin{aligned}
& b_i = a_i \oplus (1 \oplus a_{i+1}).a_{i+2} \\
\implies & a_i = b_i \oplus (1 \oplus a_{i+1}).a_{i+2} \ \text{(using } b = a \oplus x \implies a = b \oplus x\text{)} \\
& = b_i \oplus (1 \oplus b_{i+1} \oplus (1 \oplus a_{i+2}).a_{i+3}).a_{i+2} \ \text{(using } x.(1 \oplus x) = 0\text{)} \\
& = b_i \oplus (1 \oplus b_{i+1}).a_{i+2} \ (...(i)) \\
& = b_i \oplus (1 \oplus b_{i+1}).(b_{i+2} \oplus (1 \oplus b_{i+3}).a_{i+4}) \ \text{(using (i))} \\
& = b_i \oplus (1 \oplus b_{i+1}).(b_{i+2} \oplus (1 \oplus b_{i+3}).b_{i+4} \oplus (1 \oplus b_{i+3}).(1 \oplus b_i).a_{i+1}) \ \text{(using (i))} \\
& = b_i \oplus (1 \oplus b_{i+1}).(b_{i+2} \oplus (1 \oplus b_{i+3}).b_{i+4}) \oplus a_i \Pi(1 \oplus b_j) \ \text{(using (i))} \\
& = b_i \oplus (1 \oplus b_{i+1}).(b_{i+2} \oplus (1 \oplus b_{i+3}).b_{i+4}) \ \text{(using } a_i \Pi(1 \oplus b_j) = 0\text{)}
\end{aligned}$$

In the above equations $x.(1 \oplus x) = 0$ is used frequently. In the second last equation, observe that when $b_0 = b_1 = b_2 = b_3 = 0$, only then the last term matters, in which case $a_i's$ are trivially 0. So, the last term i.e. $a_i \Pi(1 \oplus b_j) = 0$. Thus, we have,

$$\begin{aligned}
\chi^{-1}(A[x,y,z]) = & A[x,y,z] \oplus (1 \oplus A[x+1,y,z]).(A[x+2,y,z] \oplus (1 \oplus A[x+3,y,z]).A[x+4,y,z]) \\
& x \in \mathbb{Z}_5, y \in \mathbb{Z}_5, z \in \mathbb{Z}_8
\end{aligned}$$

3 Inverting θ

θ is a linear tranformation.Let us define what θ is. Let $A[x,y,z]$ where $x \in \mathbb{Z}_5, y \in \mathbb{Z}_5$ and $z \in \mathbb{Z}_8$ be a bit in the cube, the transformation θ is defined by,

$$\begin{aligned}
\theta(A[x,y,z]) &= A[x,y,z] \oplus D[x,z] \\
\text{where } D[x,z] &= C[x-1,z] \oplus C[x+1,z-1] \\
\text{where } C[x,z] &= \sum_{j \in \mathbb{Z}_5} A[x,j,z]
\end{aligned}$$

Now, consider this little analysis,

$$\begin{aligned}
C'[x,z] &= \sum_{j \in \mathbb{Z}_5} \theta(A[x,j,z]) \\
&= \sum_{j \in \mathbb{Z}_5} A[x,j,z] \oplus D[x,z] \\
&= C[x,z] \oplus C[x+1,z-1] \oplus C[x-1,z] \quad ..(1)
\end{aligned}$$

Similar to equation (1), we get 40 linear equations. We can create two vectors of length 40 namely $\mathbf{c}'$ such that $\mathbf{c}'[5z+x] = C'[x,z]\ \forall x \in \mathbb{Z}_5, z \in \mathbb{Z}_8$ and $\mathbf{c}$ such that $\mathbf{c}[5z+x] = C[x,z]\ \forall x \in \mathbb{Z}_5, z \in \mathbb{Z}_8$. Let $\mathbf{M}$ be the matrix such that $\mathbf{c}' = \mathbf{M}\mathbf{c}$ to represent all the 40 linear equations. The aim is now to compute $\mathbf{M}^{-1}$. Here is the sage script we wrote for obtaining the inverse of $\mathbf{M}$.

```
def positive_mod(n,m):
    # return n % m
    ans = n % m
    if ans >= 0:
        return ans
    else:
        return m + ans
```

```
A = matrix(GF(2),40,40)
for x in range(5):
    for z in range(8):
        idx = 5*z + x
        idx1 = 5*z + positive_mod(x-1,5)
        idx2 = 5*positive_mod(z-1,8)+ positive_mod(x+1,5)
        A[idx,idx] = 1
        A[idx,idx1] = 1
        A[idx,idx2] = 1
print(A.inverse())
```

Once we find $\mathbf{M}^{-1}$ finding $C[x,z]$ given the values of $C'[x,z]$ $\forall x,z$ is trivial.

$$\begin{aligned}C[x,z] = {} & C'[x,z] \oplus C'[x+3,z] \oplus C'[x+1,z+1] \oplus C'[x+2,z+1] \oplus C'[x+3,z+1] \oplus C'[x+2,z+2] \\ & \oplus C'[x+4,z+2] \oplus C'[x,z+3] \oplus C'[x+1,z+3] \oplus C'[x+2,z+3] \oplus C'[x+3,z+3] \oplus C'[x+2,z+4] \\ & \oplus C'[x+3,z+4] \oplus C'[x,z+5] \oplus C'[x+2,z+5] \oplus C'[x,z+6] \oplus C'[x+1,z+6] \oplus C'[x+3,z+6] \\ & \oplus C'[x+4,z+6] \oplus C'[x,z+7] \oplus C'[x+1,z+7]\end{aligned}$$

$$\begin{aligned}&\theta^{-1}(A[x,y,z]) = A[x,y,z] \oplus D[x,z] \\ &\text{where } D[x,z] = C[x-1,z] \oplus C[x+1,z-1] \\ &\text{where } C[x,z] = C'[x,z] \oplus C'[x+3,z] \oplus C'[x+1,z+1] \oplus C'[x+2,z+1] \oplus C'[x+3,z+1] \oplus C'[x+2,z+2] \\ &\qquad \oplus C'[x+4,z+2] \oplus C'[x,z+3] \oplus C'[x+1,z+3] \oplus C'[x+2,z+3] \oplus C'[x+3,z+3] \oplus C'[x+2,z+4] \\ &\qquad \oplus C'[x+3,z+4] \oplus C'[x,z+5] \oplus C'[x+2,z+5] \oplus C'[x,z+6] \oplus C'[x+1,z+6] \oplus C'[x+3,z+6] \\ &\qquad \oplus C'[x+4,z+6] \oplus C'[x,z+7] \oplus C'[x+1,z+7] \\ &\text{where } C'[x,z] = \sum_{j \in \mathbb{Z}_5} A(x,j,z)\end{aligned}$$

4 Security of $F = RoR$

We have discussed how to invert χ and θ. Incerting π and ρ is fairly trivial as described in following sections.

4.1 Inverting π

We know that $\pi(A[x,y]) = A[y, 2x+3y]$, therefore $\pi^{-1}(A[x,y]) = A[3y+x, x]$.

4.2 Inverting ρ

We know that $\rho(A[x,y]) = A[x,y] << r[x,y]$, therefore $\rho^{-1}(A[x,y]) = A[x,y] >> r[x,y]$.

Thus, we now know how to compute R^{-1} (follows from 2,3,4.1,4.2) and therefore, also F^{-1}.

5 Attacks on WECCAK

5.1 Collision attack

5.1.1 Explanation

As the definition of **Collision Attack** in Section 1.3, we need to find two inputs m_1 and m_2 such that $hash(m_1) = hash(m_2)$.

We denote a message as M_i which is 184 bits long. Since the input required by WECCAK is 200 bits long, we append 16 zeroes, i.e. 0_{16} to the input making the input $(M_i||0_{16})$.

Now we apply F on the input, we get $C_i = F(M_i||0_{16})$. We keep operating F on random $M_i's$ such that $C_j[-16:] = C_k[-16:]$ $j \neq k,\ \exists j,k \leq i$, i.e. last 16 bits are same for outputs of 2 different inputs.

The last 16 bits of the output can have atmost 2^{16} values. Therefore, by the Pigeonhole principle, we can state that we require atmost $2^{16}+1$ different inputs to get a pair such that $C_j[-16:] = C_k[-16:]\ j \neq k,\ \exists j, k \leq i$.

Without loss of generality let M_1 and M_2 be two inputs for which the output coincides at last 16 bits.Now, consider the message blocks $M_1' = (M_1||C_1[:-16])$ and $M_2' = (M_2||C_2[:-16])$, both of size 368 bits and different.Now we will show that their hash will be the same.

We will consider $M_1' = (M_1||C_1[:-16])$. M_1' is 368 bits long, therefore, it will take two rounds of WECCAK. After the first round, we get C_1 as output. Before giving input to the hashing function in 2^{nd} round we xor output of the first round to the input of the second round. We have $(C_1[:-16]||0_{16}) \oplus C_1 = 0_{184}||C_1[-16:]$ where 0_{184} is a string of $0's$ 184 bits long. Therefore, input to the hash function for second round is $0_{184}||C_1[-16:]$. Similarly, for M_2' we have the input to the hash function for second round is $0_{184}||C_2[-16:]$.

Now, we have $WECCAK(M_1') = F(0_{184}||C_1[-16:])[:80]$ and $WECCAK(M_2') = F(0_{184}||C_2[-16:])[:80]$, also we have $C_1[-16:] = C_2[-16:]$ so $WECCAK(M_1') = WECCAK(M_2')$. Therefore, collision attack can be successfully executed on $WECCAK$.

The above observation establishes that WECCAK is as good as nothing towards collision attack. However, it is worthwhile to note that we don't rely on structural property of F in the collision attack. The weakness lies in having only $c = 16$ bits as capacity. This means that no matter how sophisticated a function F is, collision attack is possible in atmost $2^{16}+1$ random plaintext attacks.

5.1.2 Probability of internal collision

Let us find the probability of internal collision for any randomly selected pair (M_1, M_2), in the above description.

Lemma 1 *Let $p_k = \mathbb{P}[C_i[-16:] = k]$ for a random M_i. Let $p_1, p_2, ...p_{2^{16}}$ be the probabilities described and $\sum p_i = 1$. Minimum collision probability is achieved when $p_1 = p_2 = .. = p_{2^{16}}$.*

The proof for above is pretty simple. Probability that two random inputs produce same last 16 bits on action of F is given by $p_1^2 + p_2^2 + ... + p_{2^{16}}^2$.

$$\begin{aligned}
\{p_1, p_2, ..., p_{2^{16}}\} &= \arg\min p_1^2 + p_2^2 + ... + p_{2^{16}}^2 \\
&= \arg\min \sum p_i^2 \\
&= \arg\min \sum p_i^2 - \frac{1}{2^{32}} \\
&= \arg\min \sum p_i^2 - (\sum \frac{p_i}{2^{16}})^2 \\
&= \arg\min \sigma^2(p_i) \\
&= \{1/2^{16}, 1/2^{16}, ..., 1/2^{16}\}
\end{aligned}$$

Lemma 2 *F is most resilient to collisions described above when last 16 bits of output are produced uniformly for any random input of 184 bits.*

The structure of functions in F are in fact designed to achieve the same. More formally, the aim of designing F is to minimize its differentiability from a random oracle [1].

Lemma 3 ***Law of Large Numbers (LLN)** states that the average of the results obtained from a large number of trials should be close to the expected value and will tend to become closer to the expected value as more trials are performed.*

We will now perform an empirical analysis. We take 10^8 random plaintexts each with 184 bits to observe the variation in the last 16 bits of output of F. The code takes around a minute to execute on a PC with 8 cores and 8 GB RAM with these many plaintexts. On x axis we have the value k, of last 16 bits output of F and on y axis we have the number of times a particular value of k occurs.

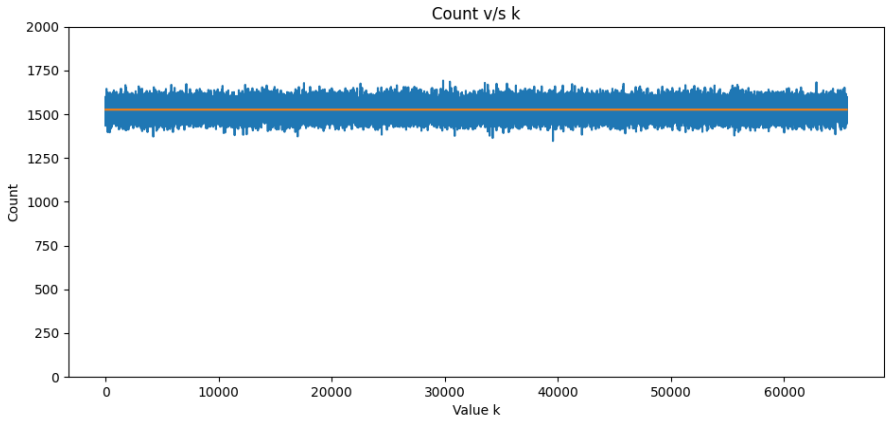

Figure 1: The count obtained for different values of $k \in [0, 2^{16} - 1]$, along with **law of large numbers** endorse the fact that value of last 16 bits is produced **uniformly** for random input on 184 bits. Orange line shows the expected count in the case of a random oracle. This property can crucially be exploited to perform pre-image attacks.

Due to uniformity of distribution, expected number of plaintext to achieve first collision is $2^{\frac{16}{2}} = 2^8$ using the birthday problem.

5.2 Preimage attack

We will crucially exploit the observation that F follows Lemma 2 empirically. The description of the attack follows.

5.2.1 Description of attack

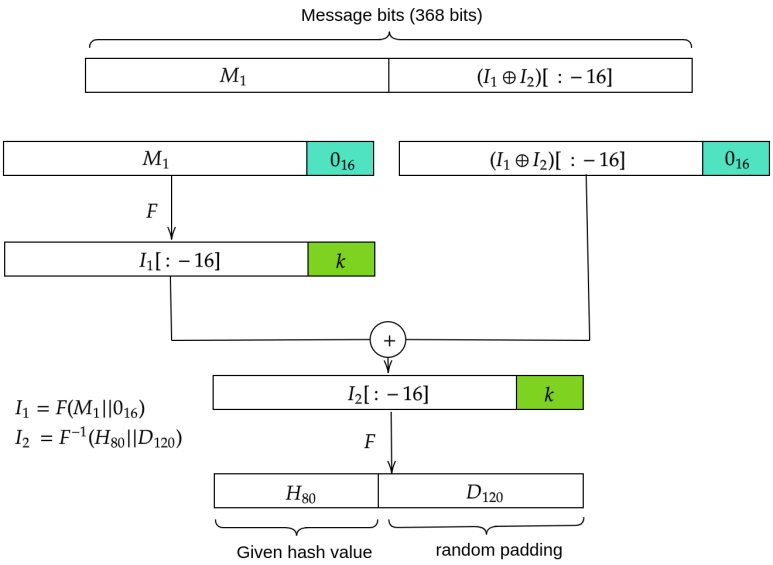

Figure 2: A schematic explaining the pre-image attack. It is remarkable to note that such an attack can be made easy by preprocessing a message M_1 for each k i.e. last 16 bits on applying F. This dictionary can be used to orchestrate an attack easily for any hash value.

According to the definition of **Preimage Attack** in 1.1, we need to show that we can get the preimage from a given hash value.

In the problem statement, the output is 80 bits long. We represent it by H_{80}

Any output of WECCAK gives output of 200 bits. To make the output 200 bits long we append 120 random bits D_{120} to H_{80} making the output 200 bits long.

To find the input corresponding to output $(H_{80}||D_{120})$, we apply F^{-1} on the output. We name the input as $I_2 = F^{-1}(H_{80}||D_{120})$. Also, we have already shown that F is invertible. We denote the last 16 bits of I_2 as k, i.e. $k = I_2[-16:0]$

We now need to find a string M_1 of length 184 such that $k = F(M_1||0_{16})[-16:]$. 0_{16} is a 16 bit string of $0's$.

Now we derive the preimage. Let the 2_{nd} half of the preimage be P_2. It is also 184 bits long. Appending 0_{16} to P_2, we have $P' = P_2||0_{16}$. Last 16 bits of I_1 and I_2 are same, i.e. k. So, we have the relation as $I_2 = I_1 \oplus P'$. After using $I_1 \oplus$ on both sides, we have $I_1 \oplus I_2 = P'$. Last 16 bits of P' will be 0_{16} since last 16 bits of I_1 and I_2 are same.Therefore, $P_2 = P'[:-16] = (I_1 \oplus I_2)[:-16]$

The complete preimage can be given by $M_1||P_2$.

We now preprocess M_1 for each value of k and store it in a dictionary. The dictionary can be constructed in around 9×10^5 random plaintexts (code attached along the writeup). This dictionary can be used to produce a preimage for any given hash value.

5.3 Second Pre-image attack

We will crucially exploit the observation that F follows Lemma 2 empirically. The description of the attack follows.

5.3.1 Description of the Attack

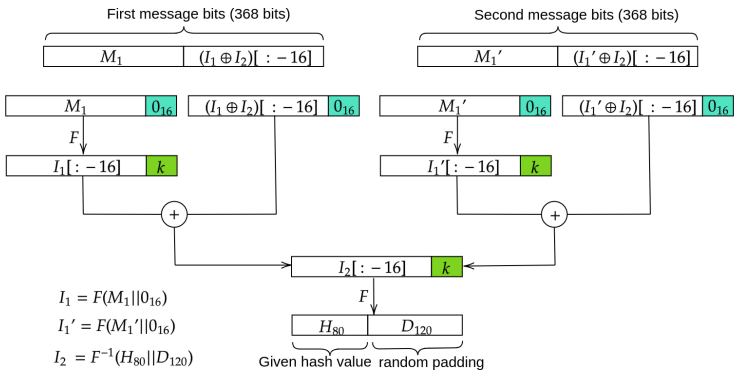

Figure 3: A schematic explaining the pre-image attack. It is remarkable to note that such an attack can be made easy by preprocessing a pair of messages $M_i's$ for each k i.e. last 16 bits on applying F. This dictionary can be used to orchestrate an attack easily for any hash value.

According to the definition of **Second Preimage Attack** in 1.2, we need to show that we can find a message which has the same hash value as that of a given message. Alternatively, we can report two preimages for given hash value.

Since we have the given message x. We take $WECCAK(x) = H_{80}$. We can apply a random padding D_{120} to this hash to get $H_{80}||D_{120}$. We can now proceed backwards in a process similar to that of the **Preimage attack**.

To find two inputs corresponding to output $(H_{80}||D_{120})$, we apply F^{-1} on the output. We name the input as $I_2 = F^{-1}(H_{80}||D_{120})$. Also, we have already shown that F is invertible. We denote the last 16 bits of I_2 as k, i.e. $k = I_2[-16:]$

We now need to find a two strings M_1 and M_1' of length 184 such that $k = F(M_1||0_{16})[-16:] = F(M_1'||0_{16})[-16:]$. 0_{16} is a 16 bit string of $0's$. Let $I_1 = F(M_1||O_{16})$ and $I_2 = F(M_1'||O_{16})$.

Now we derive two preimages. Let the 2^{nd} half of the preimages be P_1 and P_1' respectively. These are also 184 bits long. Appending 0_{16} to P_1 and P_1', we have $P_2 = P_1||0_{16}$ and $P_2' = P_1'||0_{16}$. Last 16 bits of I_1, I_1' and I_2 are same, i.e. k. So, we have the relations, $I_2 = I_1 \oplus P_2$ and $I_2 = I_1' \oplus P_2'$. Thus, we have $I_1 \oplus I_2 = P_2$ and $I_1' \oplus I_2 = P_2'$. Last 16 bits of P_2 and P_2' will be 0_{16} .Therefore, we have $P_1 = P_2[:-16] = (I_1 \oplus I_2)[:-16]$ and $P_1' = P_2'[:-16] = (I_1' \oplus I_2)[:-16]$.

The two preimages can be given by $M_1||P_1$ and $M_1'||P_1'$, each of 368 bits.

Again, another solution is to store atleast a pair of $M_i's$ for each of the 2^{16} possible values of last 16 bits of $F(M_i||0_{16})$. We wrote a code in C++ to achieve the same. We could generate such a dictionary in around $5s$ on the same PC. It required around 1.1×10^6 random plaintexts which is only slightly larger than the plaintexts required for generating a dictionary in preimage attack.With this dictionary we can produce a second preimage for any given hash value. The code can be found along with writeup.

Note: The Preimage and Second pre-image attack can be made fast by randomizing D_{120} and M_1 simultaneously until they have same last 16 bits upon applying F^{-1} and F respectively.

6 Conclusion

Construction of F ensures that the hash function is indifferent from a random oracle. As we have seen, that it is vulnerable to collision attacks, preimage attack, and second pre-image attack. Thus, WECCAK is not even remotely secure, the possible ways to make it more resilient to attacks is-

- Increase capacity $c = 16$ bits to minimize vulnerability to collisions.
- Make the rounds distinct by using a key sequence or a round dependent operation.

7 References

1. On the Indifferentiability of the Sponge Construction, KECCAK [link]
2. Law of Large Numbers (LLN) [link]
3. Birthday Problem [link]
4. Collision Attack [link]
5. Preimage Attack [link]