ENSICAEN
ÉCOLE NATIONALE SUPÉRIEURE D'INGÉNIEURS DE CAEN
& CENTRE DE RECHERCHE

SHA-3 and Midgame attacks

DELMAS Arnaud

2A Informatique – Monétique

Tutors : Slobodan Petrovic & Morgan Barbier

2014 – 2015

ENSICAEN
6, boulevard Maréchal Juin – CS 45 053 – F- 14050 Caen Cedex 4
Tél. +33 (0)2 31 45 27 50
Fax +33 (0)2 31 45 27 60

Une grande école pour réussir

Table of contents

Acknowledgements

I would like to thank our tutor M. Slobodan Petrovic, who convinced us to work on this very interesting topic, and provided a lot of useful documents as well as an outline to follow.

I am also grateful to M. Morgan Barbier, our school tutor, for his courses on cryptography and his availability, eventhough we didn't have a lot of questions due to our topic being very specific and not so technical.

Last, but not least, thanks to the ENSICAEN students I worked with during this internship : Anas Rahbi, Grégory Rabas and Adrien Genestar, as well as the Ph.D students and professors who showed up at our presentation at the Gjøvik university college : Andrii Shalaginov, Ambika Shrestha Chitrakar and especially Professor Stewart Kowalski for their insight that greatly contributed to the improvement of our work.

Introduction

The following report describes my work during a 3-month internship (from may, 5th to august, 6th 2015) I did at the Gjøvik university college[1], Norway as part of my second-year studies at the engineering school of Caen (ENSICAEN). During this internship in the information security lab, I had the chance to work on a very interesting and not so documented topic : the newest standard hash function SHA-3 and its security.
Under the supervision of professor Slobodan Petrović, I and three other students from ENSICAEN were given the mission to understand the SHA-3 algorithm construction and gather information on a recently found type of attacks that could affect its security, called midgame attack.

The Gjøvik University College

The GUC, or Høgskolen I Gjøvik, was created in 1994 in the city of Gjøvik, located approximately 100 km north of Oslo. It counts about 3400 students and 370 employees, who do research, teach or study in three different faculties : Computer Science & Media technology, Health and Economics & Management. The university truly focuses on its international perspective, with almost 40 countries represented among the students and staff, and a cooperation with 70 universities from all over the world.

Fig. 1: The Gjøvik University College main entrance

Besides, GUC also has a well-known research department with labs that are unique in Scandinavia, including Care Research, Biometrics, Forensic and the Norwegian Information Security Laboratory (NISLab), in which I had the chance to work.

The NSILab is one of the larger cyber secury groups in Europe : it mainly focuses on cryptography and forensics and counts around 50 people from different levels (professors, Ph.D, M.Sc and B.Sc).

Summary

After the discovery of collisions in SHA-1 in 2005, the National Institute of Standards and Technology (NIST) organized a 5-year competition from 2007 to 2012 to find a more secure and optimized

successor. Keccak, designed by Joan Daemen, Michaël Peeters and Guido Bertoni, won the competition out of the 51 hash function entries[2] selected for the first round. Standardized by the NIST in 2015, it is now called SHA-3. Even though SHA-3 will soon probably be one of the most used algorithms to store password hashes, it is not at the moment that well documented, and its vulnerability to midgame attacks even less, that is why working on this topic was truly appealing.

In order to try and popularize this topic, we will first talk about hash functions in general, their role, properties and constructions. Then, we are going to present Keccak construction and performances, and end with its security against classical attacks and a more advanced and recent one, a midgame attack.

1 Hash functions

1.1 Role of hash functions

Hash functions are widely used in a lot of fields of computer science, but this report will focus on the security part. One of their main utilities is to uniquely identify information, be it a password, a file – any kind of bit string, while hiding it, since it can't be decoded from the hash.

For instance, websites can't store passwords in plain text in their databases, for obvious security and anonymity reasons : they use a hash function on the password and store its hash, as shown in the figure 2 below. Whenever you log in, the password you typed in is hashed and compared with the one stored in the database. This way, even if the database is hacked, it will be impossible for the attackers to retrieve the passwords.

id	login	password
55	stellaaaaa	9717c083d7df4ca81ee56a1e01b0ddd5055133c7
56	Mizabeotut	d072e1ed70b1931c643cd0f484460297702087e6
29	behamisdy	419a2cb3fcb0b79e9e3b388325b73e63b707b379
32	Yangxinran	0869214e3d38290c6d8c4dda360478ac895a484f
33	Biver503	ad5f55c220f717ec2bf541b370e18e7e0a7d4e84

Fig. 2: Example of hashed passwords in a server database

Other common usages of hash functions are HMAC and digital signatures, that rely on hash functions to make sure of the integrity of a message and authenticate its sender, using the uniqueness and the one-way properties of such functions.

1.2 Properties

There are several properties a cryptographic hash function needs to fulfill to deserve its name. The most important ones are :

- First pre-image resistance : It is impossible to find the initial data from the hash
- Seconde pre-image resistance : Given a message and its hash, it is impossible to find a different message with the same hash
- Collision resistance : It is impossible to find two different messages with the same hash

These properties make sure that any modifications on a message has an impact on its hash, and that one cannot find the initial message given its hash.

1.3 Constructions

With compression functions

The first possible construction is based on one-way compression functions : the message is split in blocks with a padding if needed and transformed with such a function to produce a hash of a given

length. Many known hash functions use such a construction : MD5, SHA-1, SHA-2...

There are a lot of possible constructions with compression functions : Davis-Meyer, Merkle–Damgård...

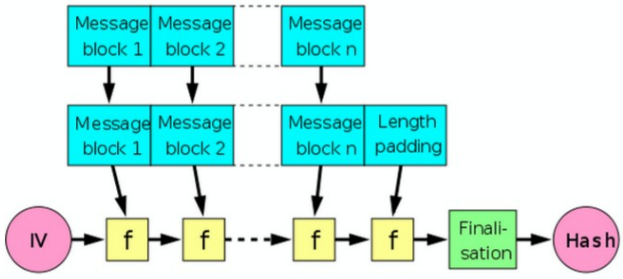


Fig. 3: The Merkle-Damgård construction

Sponge construction

The main characteristic of this construction is to take an input of any length and produce an output of a choosen length. Keccak is based on a sponge construction, which gave him an edge against his competitors.

Figure 4 shows how a sponge construction works. Such a function has three components :

- a state, composed of a bitrate r and a capacity c
- a permutation function f
- a padding function p

...and two phases :

- the absorbing phase : blocks of r bits are XORed with the bits of the state and go through f until all the blocks are absorbed
- the squeezing phase : r-sized output blocks are generated from the state. If more are needed, f is applied on the state again

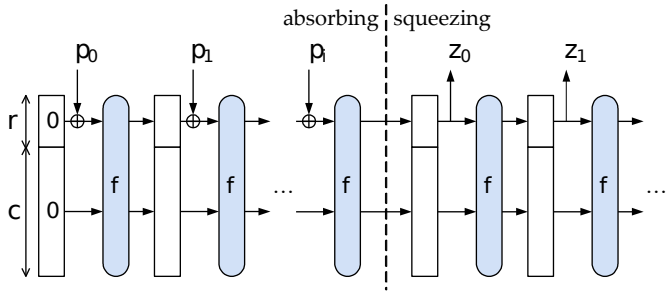


Fig. 4: The two phases of a sponge construction. P are the input and Z the output blocks

2 About Keccak

2.1 Construction

Keccak uses a sponge construction. Its block permutation function consists of a composition of five individual functions in a 24 rounds loop, and its state is a 5x5x2^l matrix, with l = 6 in the SHA-3 version.

$$R = \iota \circ \chi \circ \pi \circ \rho \circ \theta \qquad (1)$$

We will now review each of the block permutation functions. You can find a code-based analysis of these functions from the C source code of SHA-3 in the appendice 1 at the end of the report.

2.2 The round function R

Vocabulary

The state being a 3-dimensional 5x5x64 matrix, the Figure 5 below illustrates the vocabulary that will be used in this report.

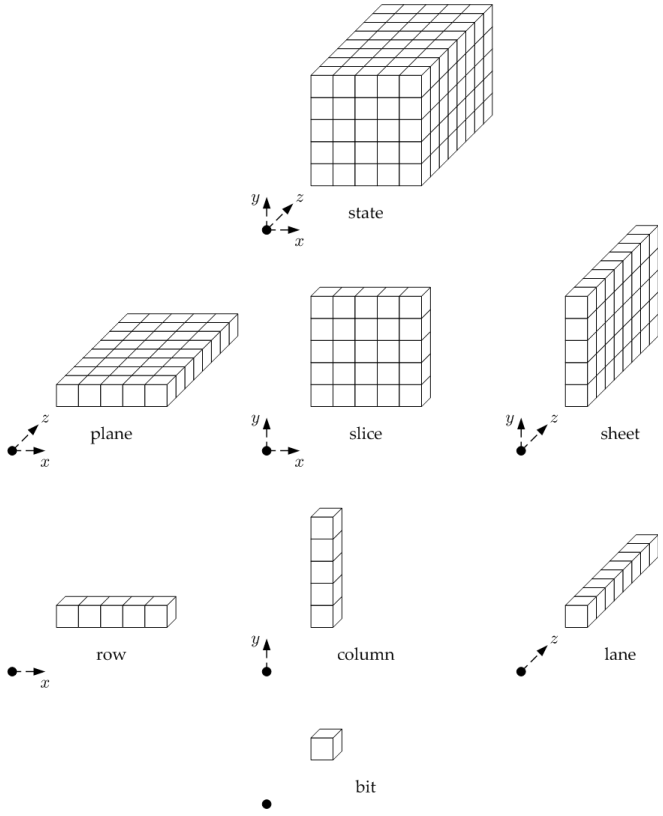


Fig. 5: Vocabulary illustration of the state matrix

Theta

Theta is the first function of Keccak. Its goal is to mix layers in the state matrix using a simple XOR between a bit in position (x, y, z) and the XOR of the bits in the column on its right in (x+1, ., z) and the one on its left in the slice below, in (x-1, ., z+1).

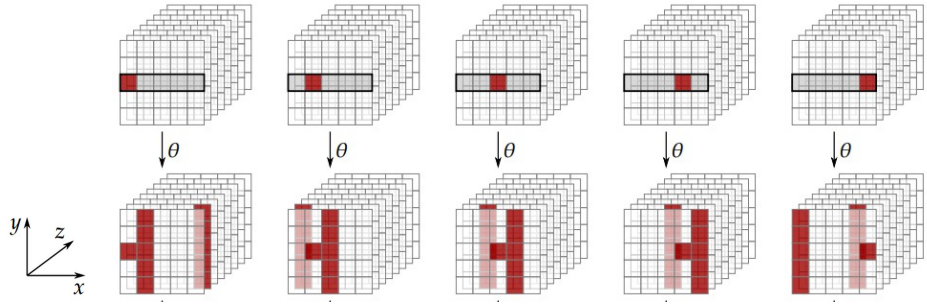


Fig. 6: Example of Theta propagation. Red squares are the bits taken into account (called 'active' bits) by Theta.

Rho & Pi

Rho and Pi respectively translates bits in z-direction by a constant value and permutes bits in a slice. Rho results in an inter-slice dispersion, while Pi's purpose is to disturb the horizontal and vertical alignment, as shown in the figures 7 & 8 below.

Equations (2) and (3) describe Rho and Pi :

$$state[x][y][z] \leftarrow state[x][y][z-(i+1)(i+2)/2] \quad (2)$$

$$state[3x+2y][x] = state[x][y] \quad (3)$$

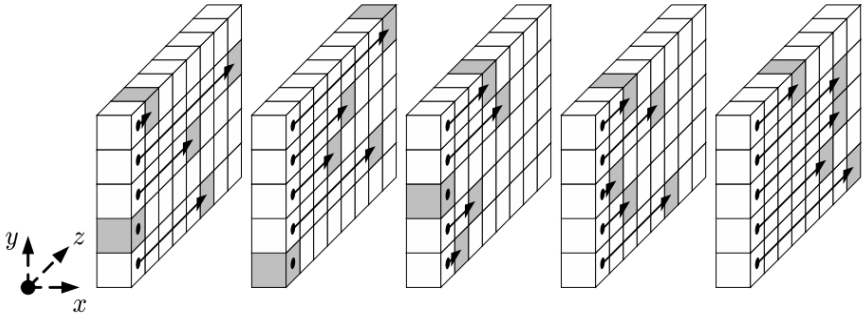


Fig. 7: Rho interslice dispersion

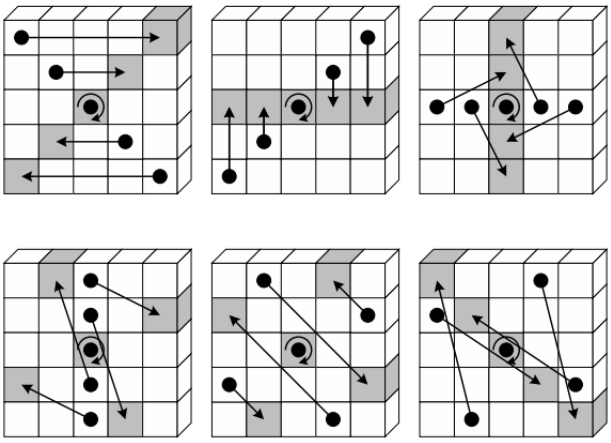


Fig. 8: Pi permutations within a slice

Chi

Chi is a bitwise combination along rows, according to substitution boxes shown in figure 9. It is the only non-linear operation in Keccak. Its formula is described in (4).

$$state[x][y][z] \leftarrow state[x][y][z] \oplus (\neg state[x][y+1][z] \wedge state[x][y+2][z]) \quad (4)$$

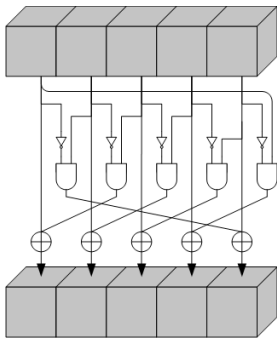


Fig. 9: Substitution boxes of Chi

Iota

This last operation consists of XORing lanes with a round constant to break the symmetry of the other functions.

It is a very important part of the algorithm : without Iota, there would be invariance to translations in z-direction, all rounds of the loop would be identical and there would be fixed points. All of these could be exploited by cryptographic attacks.

Summary

θ : Mixes layers

ρ : Inter-slice transpositions

π : Intra-slice transpositions to disturb alignment

χ : Non-linear operation

ι : Adds round-dependant constants

2.3 Performances

As we saw in the previous part, Keccak is very flexible, and composed with simple operations to implement in both software and hardware wise. The algorithm truly shines in this field and it is one of the reasons the NIST chose it.

Performances in software

Keccak is faster than his predecessor SHA-2 and md5 on some processors, as shown in figure 10.

C/b	Algo	Strength
4.79	keccakc256treed2	128
4.98	md5 broken!	64
5.89	keccakc512treed2	256
6.09	sha1 broken!	80
8.25	keccakc256	128
10.02	keccakc512	256
13.73	sha512	256
21.66	sha256	128

Fig. 10: Performances in software (measured in Cycles/bytes on an AMD Buldozer) of well-known hash functions.

Furthermore, as Keccak is meant to be implemented on all sorts of plateforms, we can look at its performances on a microcontroller : the AVR Atmel Atmenga1248P.

On this kind of chips, the memory usage is way more important than speed. Figure 11 shows a comparison of speed and memory with other hash functions.

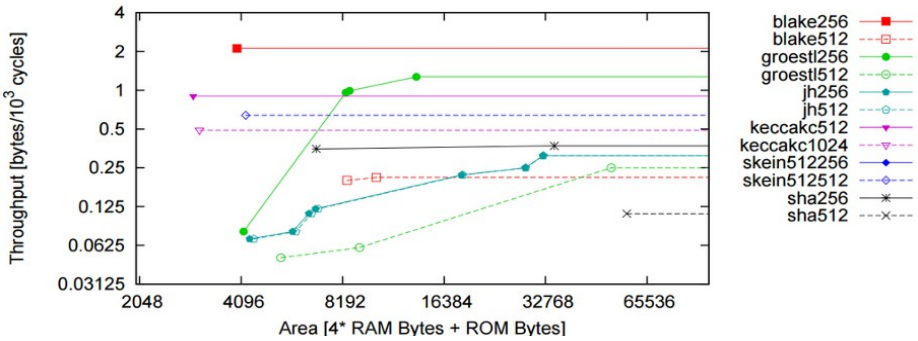


Fig. 11: Performances in software measured in speed and memory usage on an AVR Atmel Atmenga1248P of well-known hash functions.

As we can see on the figure above, Keccak is not the fastest on this plateform, but its memory usage is the smallest which is perfect for these types of microcontrollers.

The two examples above show Keccak's flexibily due to its sponge construction and its round function, as it adapts very well to any platforms whether we value its speed or its performances memory-wise.

Performances in hardware

What is impressive about Keccak is its performances in hardware. Its construction allows it to be way better on hardware devices than the other hash functions. This comes from the fact that its internal state isn't that big for modern technologies, and its round function is composed with very simple and fast operations : some XOR, AND and permutations.

Figure 12 shows that Keccak has a way better throughput than other hash functions such as SHA-2 or BLAKE, and is decent in memory usage.

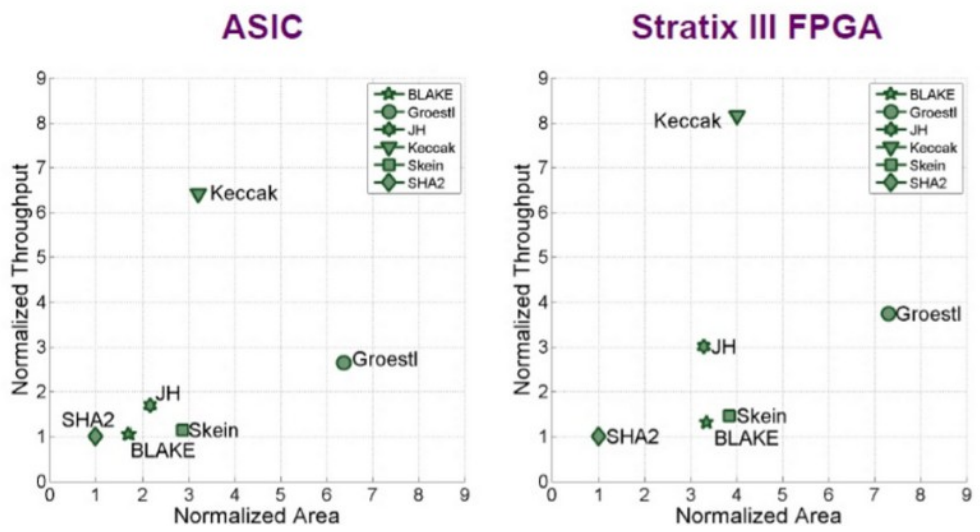


Fig. 12: Memory usage and throughput comparison between hash functions on two different hardware configurations.

3 Security of Keccak

3.1 Classical attacks

Collisions

In SHA-3's sponge construction, the parameter c is 512, which guarantees a security strength 512/2 = 256 : that is way enough to make sure finding collisions is impossible.

The permutation function R inside Keccak is proven secure against generic attacks, but could still be exploitable by specific designed attacks.

On a side note, one of Keccak's strength is that you can choose the bitrate and the capacity of its sponge construction to fit your security requirements. Keccak's authors made a special page to inform about r and c for a desired resistance, as shown in figure 13.

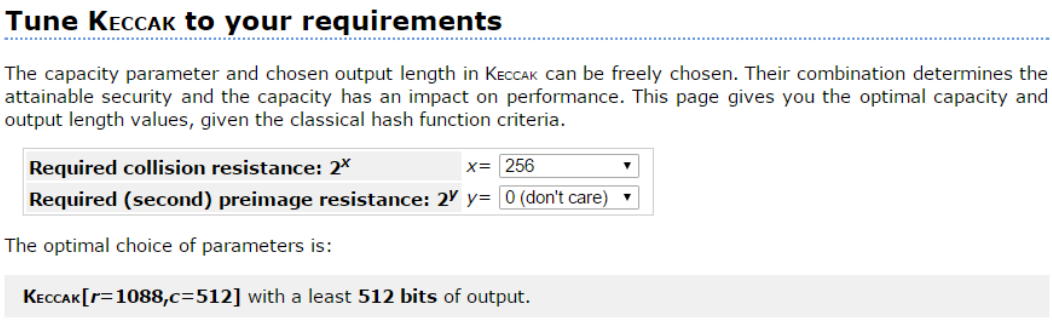


Fig. 13: From http://keccak.noekeon.org/tune.html
On this page, you can specify the security you want and get the optimal parameters to match it with the best speed possible

Rotational & slide attacks

Rotational attacks is a generic name for the attacks against algorithms that only use modular additions (AND), rotations and XOR. All these operations preserve rotational relations : that's what this attack is based on.

Slide attacks are meant to break ciphers that rely on a single function and rounds.

Both these attacks are nullified by the Chi and Iota operations which respectively add a non-linear step and a round constant in the algorithm.

Side-channel attacks

These attacks rely on leakage from an external point of view : power, radiations, time, temperature... with different messages and compare the results to get information about the internal state.

Most side-channel attack related papers focus on the first step of Keccak, the Theta function, which all rows are pre-calculated and stored (see Appendice 1 for more information). A power analysis of this part has shown good results[3].

A Differential Power Analysis (DPA) can also exploit the Chi operation[4], more precisely by monitoring the consumption of bit b0 shown in figure 14, but these kind of attacks are well-known and protection can easily be added in the implementation at a lesser cost to prevent them[5].

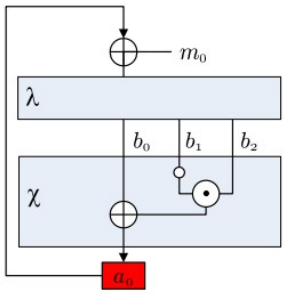


Fig. 14: In this figure, λ = π ∘ ρ ∘ ϑ and b0, b1, b2 are the first three bits of the χ S-box

Still, it is not enough for these attacks to be called effective and reliable.

3.2 Advanced Persistent Threat and Midgame attacks

Definition of an Advanced Persistent Threat (APT)

An APT is a stealth attack aiming at staying hidden over a long period of time. It is a sophisticated attack that makes use of all kind of hacking techniques to achieve its goal : penetrate into a system, monitor it and extract datas while remaining secret as long as possible. The designation APT is very recent : the term was first used in 2006. Figure 15 shows the life cycle of an APT and illustrates how organized and sophisticated it is.

The first APT was called Titan Rain : a series of attacks starting in 2003 and ongoing for about 3 years against United States defense contractors including NASA. They were belived to come from Chinese military hackers. Another well-known example is the highly specialized version of the worm Stuxnet, discovered in 2010, that stayed hidden several months to monitor and give access to industrial systems.

APT are hard to set up : they mostly come from big companies or governements.

Fig. 15: The life cycle of an APT compared to other threats

Edward Snowden, in his revelations, tells that the NSA itself might be performing Advanced Persistent Threat in our computers.

Midgame attacks and Keccak

Once an APT is set up, an attacker can perform a midgame attack. This attack consists of dumping and analyzing the whole memory of a program or a system.
This attack mainly affects cryptography, since it can pause an algorithm in the middle of a computation, dump its memory and use it to recover the secret key. A lot of algorithms are broken by midgame attacks : 3DES, AES...
A way to prevent them is using split architectures : pre-process and post-process the algorithm on a safe environment (HSM, computer of the user...), while the rest of the work can be done on an unsafe environment (such as a cloud) from a midgame state without knowing the key, from which it is impossible to recover vital information even with a memory dump. This technique is called Multi-party computation (MPC).

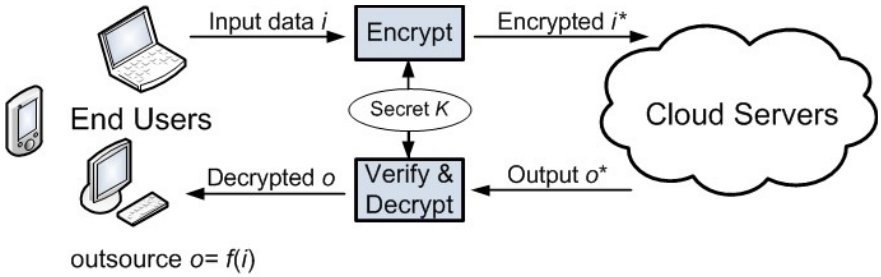


Fig. 16: A Multi-party computation where the workload is done in a cloud

Moti Yung, an Israeli-American cryptographer and employed at Google, was the first one to talk about SHA-3 vulnerability against midgame attacks in a presentation at the Crypto 2012 rump session[6]. He explains that out of the five SHA-3 finalists chosen by NIST, only Keccak is vulnerable to these attacks.
The reason is, while a lot of constructions such as Merkle-Damgård, SHA-1 or SHA-2 use a non-invertible function, Keccak uses a sponge construction with invertible permutation function. For a sponge construction to be one-way, the capacity c must remain hidden, which is not the case since the internal state is leaked when a midgame attack is performed. If you can access any iteration of its internal state, Keccak becomes invertible : any HMAC-Keccak usage is then untrustworthy.

This kind of “midgame thinking” is pretty recent : it emerged from the development of secure MPC and particularly cloud MPC : if a cloud performing crypto algorithms vulnerable to midgame attack is hacked, all the keys used could be recovered and the information decoded which would be a disaster.

Conclusion

This internship was very enriching : the topic suggested by our tutor was very interesting to work on, since it is relatively new and not so well documented. We had to search information on our own, with the help of our tutors and our knowledge, which was tedious but truly rewarding. Learning about Keccak not only allowed us to find out about hash function implementations and properties in a practical way, but also made us learn about all kind of attacks, some of them very recent.

As someone who wants to work in computer security, I gained a huge experience from this internship, and I am glad that I could achieve this work in such a good research department. While my second-year project consisted in the implementation of the E0 cryptosystem, this internship covered another very important part of modern cryptography and improved my knowledge in this field in a huge way.

I hope I will have the chance to keep working on cryptography during my next internship or project since it is a field I'm definitely interested in.

4 BIBLIOGRAPHIC REFERENCES

[1] https://www.hig.no

[2] http://csrc.nist.gov/groups/ST/hash/sha-3/Round1/submissions_rnd1.html

[3] https://eprint.iacr.org/2015/411.pdf

[4] http://keccak.noekeon.org/Keccak-slides-at-COSADE-Mar2013.pdf

[5] http://keccak.noekeon.org/NoteSideChannelAttacks.pdf

[6] http://crypto.2012.rump.cr.yp.to/

Other references

Most of the figures concerning Keccak construction were found in the following presentations :

http://keccak.noekeon.org/Keccak-slides-at-Eurocrypt-May2013.pdf (en)

http://keccak.noekeon.org/Keccak-slides-at-CCA-Jan2013.pdf (fr)

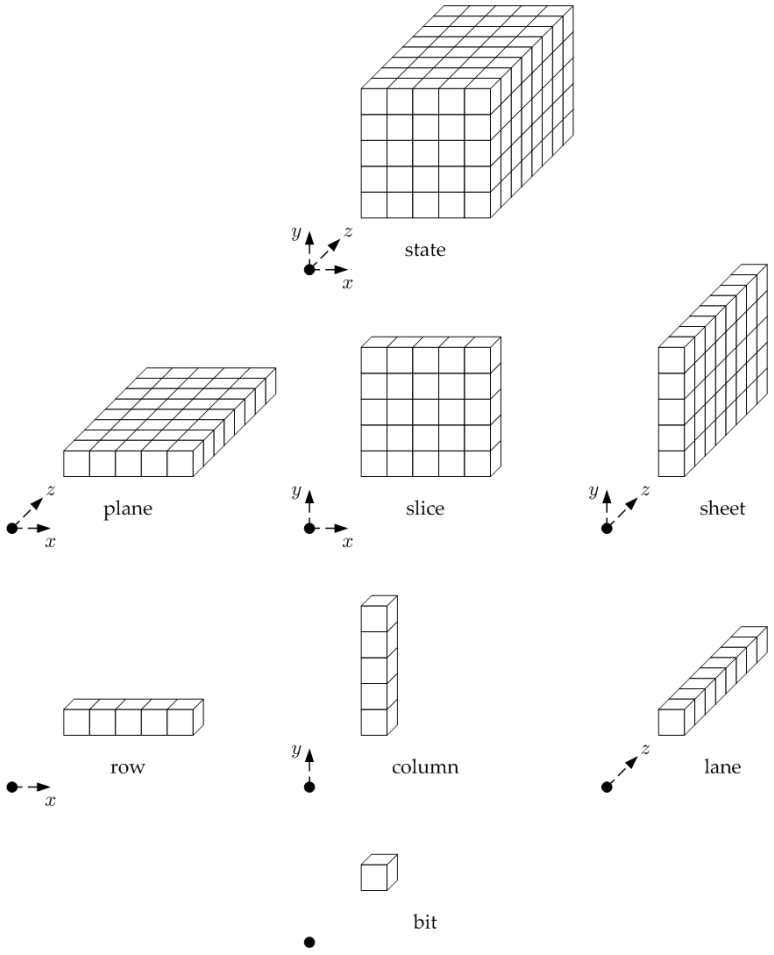
The source code of Keccak can be found on the official website : http://keccak.noekeon.org/

Appendix 1 :

C Implementation of the Keccak round function

http://keccak.noekeon.org/KeccakReferenceAndOptimized-3.2.zip
Keccak-compact.c

Vocabulary



Moreover, what we will call “state array” refers to the state matrix in memory, declared as “tKeccakLane state[25]”.

Data structure

The main component of Keccak is its state. For optimization purpose, Keccak uses a 1-dimension array of 25 « tKeccakLane », which is a 64 bits unsigned int, as the 5x5x64 state matrix. This seems to be the best option, as the linear operations used such as XOR are performed bit-to-bit by default and translation in z-direction can be performed with a simple rotation of a lane.

Theta

This is the first function of Keccak. Its goal is to mix layers using a simple XOR between a bit in position (x, y, z) and the XOR of the bits in the column on its left in (x-1, ., z) and the one on its right in the slice below, in (x+1, ., z-1).

Code explanation :

In each round :

a) Compute and store the XOR of all 5 sheets in an array (BC[5])

b) Store the XOR between the sheet on the left and the sheet on the right after a rotation to the left so that it is done with the slice below

c) XOR the result with the bits in the current column.

d) Go the next column

Rho & Pi

Rho and Pi are performed simultaneously in the same loop. They consists respectively in translating bits in z-direction and permuting bits in a slice.

Considering that there is no 0 in the array KeccakF_PiLane and that the fixed point of Pi is the element in the center (2, 2, .), we figured out that this element was at the position 0 in the state array. From there, given the figures and some explanations in the reference, we were able to deduce the order in the state array.

Here is an illustration showing how the state is stored in the array, in relation with the cubic figures provided in the reference :

13	14	10	11	12
8	9	5	6	7
3	4	0	1	2
23	24	20	21	22
18	19	15	16	17

Relation between the state figures in the reference and the state array in the source code.

Numbers in each case are the corresponding position in the state array.

This was the key to understanding this part of the algorithm, especially the values in KeccakF_PiLane and KeccakF_RotationConstants as they look totally random if you think that the state array is arranged in the natural order (i.e. The bottom-left or top-left of the matrix is at the position 0 etc.).

We can now move on and fully understand the code itself.

Developers chose to store Rho and Pi constants in two arrays instead of computing them dynamically in the loop, which is understandable considering the number of iterations of the round function.

a) First, the lane at the position x=1 (the lane on the right of the center element in the figures) is stored in a temporary variable temp

b) Then, a 24-iterations loop starts, since Rho consists in 6x4 operations

c) The lane that is going to be modified by Rho is saved in BC[0]

d) This lane is replaced with the lane transformed by the Rho translations. Once again these translations are done with the ROL(lane, n) macro which is a simple rotation to the left of the lane n-times

e) BC[0] is transfered into the temp variable so that the algorithm can continue as if Rho and Pi were done one after the other, in two separate loops

Chi and Iota

These two functions are pretty simple. The first one is just a Substitution-box applied on each row of the state array. Iota is a basic XOR between the center lane and a constant stored in the array KeccakF_RoundConstants depending on the round.