BIL470 – HOMEWORK REPORT

CANBERK ARICI - 171044062

***** I also send my report as pdf in case of any slippage that may occur in the docx file.**

What is Sparkle?

It is a family of cryptographic permutations based on an ARX design. Sparkle is named after the block cipher Sparx, to which it is closely related. Sparkle is just a Sparx instance with a larger block size and a fixed key, thus the name: *SPARx, but Key Less*.

What Are Esch and Schwaemm?

Schwaemm is a light-weight cryptography method and provides confidentiality, integrity and authentication, while Esch provides a hashing method that is preimage and collision resistant. Both methods use a sponge construction using a cryptographic permutation (as used in SHA-3). Esch256 (Efficient, Sponge-based, and Cheap Hashing) implements the hashing method for a 256-bit hash, and has a block size of 16 bytes, a security level of 128 bits and a data limit up to 2^{132}.

What is Elephant?

The Elephant authenticated encryption technique is introduced. Elephant's mode is a nonce-based encrypt-then-MAC design, in which message encryption is done in counter mode and message authentication is done with a variation of the protected counter sum MAC function. Internally, both modes employ a cryptographic permutation disguised by LFSRs, similar to Granger's masked Even-Mansour architecture. Elephant's parallelizability eliminates the requirement for big permutation instances: we may use as little as 160-bit permutations while still meeting the security standards set out by the NIST lightweight request. The Elephant plan is broken down into three parts:

1- Dumbo: Elephant-Spongent-π: The minimal permutation size required by the security analysis is met by this instance.

2- Jumbo: Elephant-Spongent-π: This is a little more conservative version of Elephant: it uses the same permutation family as the original, but achieves 127-bit security under the same online complexity criteria.

3- Delirium: Elephant-Keccak-f :Although it works quite well in hardware, this variation is geared mainly for software use. Elephant instantiated with Keccak-f also provides 127-bit security, with an online complexity constraint of roughly 270 blocks.

<u>IMPLEMENTATION COMPARISON</u>

SPARKLE

- **Alzette**
  The ARX-box Alzette is a key component of Sparkle, and as such, it was created with high security constraints in mind, as well as efficient implementation. On ARM processors, operations of the type $z \leftarrow x$ <op> $(y \lll n)$ may be completed in a single clock cycle with a single instruction, regardless of rotation distance. Regardless of the rotation distance, a single clock cycle. Alzette works with two 32-bit words of data and an additional 32-bit constant value. This enables full in-register processing in AVR, MSP, and ARM architectures, the latter of which can carry at least four Alzette instances fully in registers. As a result, load-store overheads are reduced, and the permutation performs better. The consistency of operations across branches, which implies that each branch performs the same sequence of instructions, enables one to focus on short code size or architectures with more registers, or to run two or more branches to take advantage of instruction pipelining. The use of Single Instruction Multiple Data (SIMD) instructions allows for some parallelism due to the consistency of operations. Furthermore, the state's modest size makes it compatible with most major SIMD engines, including ARM's NEON and Intel's SSE or AVX.

- **Linear Layer**
  Although the branch permutation may be implemented at the end of the linear layer, the combination of the two operations has a unique cycle, allowing it to be implemented in a single loop.

- **Parameterized Implementations**
  This makes it easier to write macro-based code that builds binaries for a given instance while still providing support for all instances of the algorithm. It's also simple to accomplish and contributes to a short code size.

- **Hardware Implementation**
  A typical Ripple-Carry Adder (RCA) made of 32 Full Adder (FA) cells is the simplest type of a 32-bit adder. Although RCAs are space-saving, their delay grows linearly with the adder's bit length. Each round of a Sparkle permutation may be implemented as a completely combinatorial circuit, which can be completed in a single clock cycle, resulting in a high-throughput implementation. The ARX-box Alzette may be created once and reused several times to run the round function, which reduces the area overhead. As a result, each round may be completed in nb clock cycles, resulting in decreased area overhead but increased latency.

- **Protection against Side-Channel Attacks**
  Side-channel attacks, particularly Differential Power Analysis, are common in straightforward implementations of symmetric cryptographic algorithms like Schwaemm (DPA). Because the Schwaemm standard does not include any conditional statements that rely on secret data, timing attacks and traditional Simple Power Analysis (SPA) attacks are of little concern. Masking, which may be implemented in both hardware and software, is a well-known and widely-used countermeasure against DPA attacks. To decorrelate the sensitive data of the algorithm from the data that is actually processed on the device, masking seeks to disguise every key-dependent variable with a random value called mask.

The masks must be changed from one form to the other without causing any leakage since Schwaemm is an ARX design that incorporates arithmetic and Boolean operations.Schwaemm takes use of a large body of research on ARX masking strategies and can successfully and efficiently defend against DPA attacks.

ELEPHANT

- Elephant supports a high level of parallelism. This makes it simple to trade off space for throughput in the hardware-oriented Elephant versions (Dumbo and Jumbo). The 200-bit Elephant variation is primarily aimed towards (embedded) software, although the same observations about hard-ware implementations apply, as proven by a hash function based on the 200-bit Keccak permutation that requires just 2520 GE. Parallelism can be used in software implementations of the 200-bit Elephant (Delirium).

- Many blocks can be processed concurrently if multiple cores are available, however this is only practical for large messages. On processors with word sizes greater than 16 bits, the available parallelism allows the implementation to be made more efficient by merging two or more calls to the Keccak permutation. The same method may be used to get even bigger speedups on mid- and high-end CPUs using SIMD instructions.

- By design, the masking system is continuous time. The Spongent and Keccak permutations are the same. Furthermore, all Elephant variations are well-suited to Boolean masking techniques like threshold implementations. Additional Elephant improvements are possible in a few specialized use-cases. For fixed-length messages, the contribution of the mask values to the tag can be precalculated. It's feasible to precompute the contribution of one or more blocks of connected data to the tag if they're static.

SECURITY ANALYSIS COMPARISON

SPARKLE

- **ESCH**
  With regard to collision resistance, preimage resistance, and second preimage resistance, we claim that Esch256 and Esch384 provide a security level of c/2 bits, where c is both the capacity and digest size. The security against length-extension attacks is covered by our claim. A $2^{c/2}$ processed block data limit is set. At most $2^{c/2}$ executions of the underlying permutation or its inverse are required for an attack c that violates the preceding security claim.

- **SCHWAEMM**
  For both secrecy and integrity, the Beetle mode of operation provides a security level of min (r, (r + c)/2, c-log2 (r), where r is the rate and c is the capacity. Schwaemm192-192 is claimed to have a security level of 184 bits, allowing the attacker to process a total of 2^{68} bytes of data with a single key. An attack that breaks this security claim takes at most 2^{184} executions of the underlying permutation or its inverse and requires at most 2^{68} blocks of data. Similarly, Schwaemm256-128 is claimed to have a security level of 120 bits, allowing the attacker to process no more than 2^{68} bytes of data with a single key. The security level of Schwaemm128-128 is reported to be 120 bits, meaning the attacker can only process 2^{68} bytes of data with a single key. Finally, Schwaemm256-256 is said to have a security level of 248 bits, allowing the attacker to process up to 2^{133} bytes of data with a single key.

- **Truncated Trails in Sparkle**
  If the weight of the reduced differential is strictly less than the number of inactive words in the output, it is considered to be effective. The longest effective truncated differential trail for Sparkle256 is two steps long and has a weight of 0.The longest effective truncated differential trail covers just one step and has weight 0 when the input difference is restricted to the left branches exclusively. The longest effective truncated differential trail for Sparkle384 is similarly two steps long and has a weight of 0. It applies to the Schwaemm256-128 setting as well. The longest successful truncated differential trail for Sparkle512 is three steps long and weighs one pound. In general, shortened trails pose no security risk to any Sparkle version.

- **Boomerang Attacks**
  Boomerang attacks appear to be rather hazardous on the ARX-box level. If we have two differentials with probability 2^{-6} for one ARX-box, we may build a Boomerang differential across two ARX-boxes with probability 2^{-24}, which is higher than the upper constraint on the maximum predicted differential trail probability of 2^{-32}. Classic Boomerang attacks do not appear to be a concern for the entire variation, thankfully.

- **Zero-Correlation Attacks**
  Security is assessed against truncated zero-correlation distinguishers using the same technique as for impossible differentials, with the transpose of the inverse of the linear layer taken into account. The longest distinguisher we discovered in that approach includes four stages for all Sparkle versions.

ELEPHANT

- **Dumbo: 160-Bit Elephant**
  The proof below shows that Dumbo is secure in the random permutation model.

  Proposition B.4. Let n = 160. Let $\phi 1 : \{0, 1\}^{160} \rightarrow \{0, 1\}^{160}$ be the LSFR given in (3), and $\phi 2 = \phi 1 \oplus$ id. The tweak space $T = T1 \times T2 = \{0, 1, . . . , 2^{154} \} \times \{0, 1, 2\}$ is 2^{-n} -proper with respect to ($\phi 1$, $\phi 2$).

  Proof: Let V be the 160 × 160 matrix over F2 that represents $\phi 1$. ϕ_1^i (L) = $V^i \cdot L$ is 2^{-n} -proper for $i \in \{0, . . . 2^n - 2\}$ if the minimal polynomial of V is primitive and of degree n. Computation show that the polynomial below is irreducible and primitive.

  $p(x) = x^{160} + x^{136} + x^{83} + x^{53} + 1$

  Next, let = $\log_x (x + 1)$ in the field F2 [x]/p(x). We have to show that $\phi_b^2 \circ \phi_a^1 (L) = (V + I)^b \cdot V^a \cdot L = V^{l-b} \cdot V^a \cdot L$ is unique for any distinct set of tweaks. Computation gives the following values for l and 2l:
  $l = 742800116542094474882643562714650758474536684889 \approx 2^{159.02}$,
  $2l = 240985957532860315616022927130184972931408268O3 \approx 2^{154.08}$.

  When we examine that $b \in \{0, 1, 2\}$ separates the tweak space into three sets, the smallest difference is between b = 0 and b = 2, which is more than 2^{154}. As a result, establishing that $0 \le a \le 2^{154}$, ensures that for every two different (a, b), (a', b') $\in \{ 0, 1,..., 2^{154}\} x \{0, 1, 2\}$, $\phi_2^b \circ \phi_1^a$!= $\phi_b^{2'} \circ \phi_a^{1'}$.

- **Jumbo: 176-Bit Elephant**
  The proof below shows that Jumbo is secure in the random permutation model.
  Proposition B.6. Let n = 176. Let $\phi 1 : \{0, 1\}^{176} \rightarrow \{0, 1\}^{176}$ be the LSFR, and $\phi 2 = \phi 1 \oplus$ id. The tweak space $T = T1 \times T2 = \{0, 1, . . . , 2^{173} \} \times \{0, 1, 2\}$ is 2^{-n} -proper with respect to ($\phi 1$, $\phi 2$).

  The 176 × 176 matrix V that represents $\phi 1$, the corresponding polynomial is irreducible and primitive.

  $p(x) = x^{176} + x^{154} + x^{135} + x^{19} + 1$

  The discrete logarithm $l = \log_x (x + 1)$ in the field F2 [x]/p(x) and its related 2l are computed as given below
  $l = 188813761514037867774814634320294502941004615622206990 \approx 2^{173.66}$,
  $2l = 377627523028075735549629268640589005882009231244413980 \approx 2^{174.66}$.

  The lowest difference between set b = 0 and set b = 1, or b= 1 and b = 2, which is larger than 2^{173}, is between set b = 0 and set b = 1, or b= 1 and b = 2. As a result, by ensuring that $0 \le a \le 2^{173}$, we may have any two unique values for (a, b), (a',b') $\in \{0, 1, . . . , 2173 \} \times \{0, 1, 2\}$, $\phi_2^b \circ \phi_1^a$!= $\phi_b^{2'} \circ \phi_a^{1'}$.

- **Delirium: 200-Bit Elephant**

  Proposition B.8. Let n = 176. Let $\phi1 : \{0, 1\}^{200} \rightarrow \{0, 1\}^{200}$ be the LSFR, and $\phi2 = \phi1 \oplus id$. The tweak space $T = T1 \times T2 = \{0, 1, . . . , 2^{197}\} \times \{0, 1, 2\}$ is 2^{-n} -proper with respect to ($\phi1$, $\phi2$).

  The 176 × 176 matrix V that represents $\phi1$, the corresponding polynomial is irreducible and primitive.

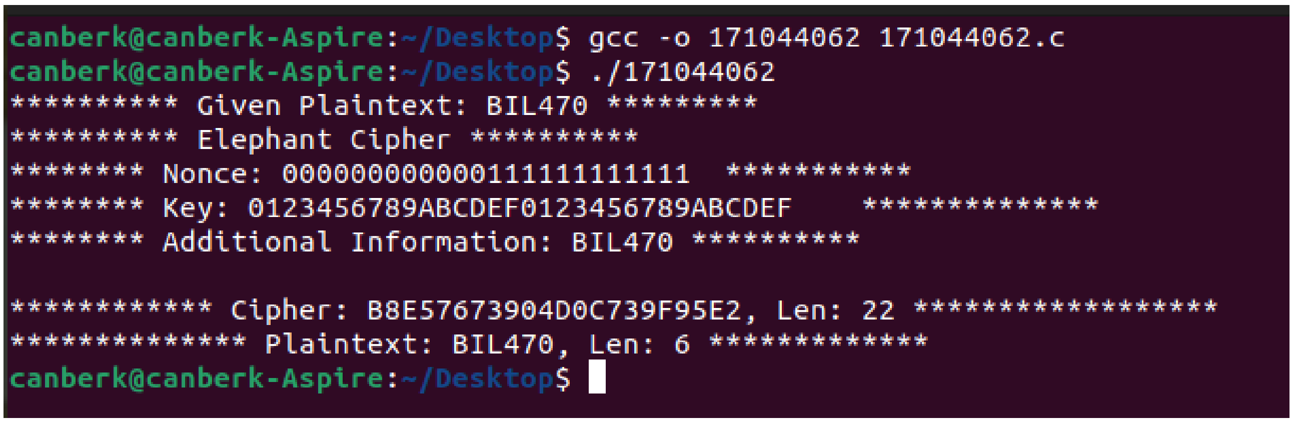
  $$p(x) = x^{200} + x^{93} + x^{91} + x^{82} + x^{78} + x^{71} + x^{69} + x^{67} + x^{65} + x^{60} + x^{52} + x^{49} + x^{47} + x^{41} + x^{39} + x^{38} + x^{30} + x^{27} + x^{26} + x^{25} + x^{23} + x^{21} + x^{19} + x^{17} + x^{16} + x^{15} + x^{13} + 1$$

  The discrete logarithm $l = \log_x (x + 1)$ in the field F2 [x]/p(x) and its related 2l are computed as given below

  l = 692180606625676931900534627786122994390018641930530681719698 $\approx 2^{198.78}$,
  2l = 1384361213251353863801069255572245988780037283861061363439396 $\approx 2^{199.78}$.

  The lowest difference between set b = 0 and b = 2, which is larger than 2^{197}, is between set b = 0 and set b = 1, or b= 1 and b = 2. As a result, by ensuring that $0 \le a \le 2^{197}$, we may have any two unique values for (a, b), (a’,b’) $\in \{0, 1, . . . , 2^{197}\} \times \{0, 1, 2\}$,
  $\phi_2^{b} \circ \phi_1^{a}$!= $\phi_b^{2'} \circ \phi_a^{1'}$.

I could implement only Elephant Algorithm and here is Elephant Algorithm’s output:



```
canberk@canberk-Aspire:~/Desktop$ gcc -o 171044062 171044062.c
canberk@canberk-Aspire:~/Desktop$ ./171044062
********** Given Plaintext: BIL470 *********
********** Elephant Cipher **********
******** Nonce: 000000000000111111111111  ***********
******** Key: 0123456789ABCDEF0123456789ABCDEF    **************
******** Additional Information: BIL470 **********

************ Cipher: B8E57673904D0C739F95E2, Len: 22 ******************
************** Plaintext: BIL470, Len: 6 *************
canberk@canberk-Aspire:~/Desktop$ 
```