Shamash *(and Shamashash)* (version 1)

Designers and submitters: Daniel Penazzi and Miguel Montes
Daniel Penazzi:
`penazzi@famaf.unc.edu.ar`
Famaf-Universidad Nacional de Córdoba. CIEM.

Miguel Montes:
UNDEF CRUC-IUA
`mmontes@iua.edu.ar`

Contents

Chapter 1

Specification

1.1 Parameters

Shamash is an authenticated cipher with 128 bit key and 128 bit nonce.

The inputs to authenticated encryption are a plaintext P, associated data A, a public message number (nonce) N, and a key K. As stated above, N and K are 128-bits each. N must be a nonce, i.e. the same N cannot be used with the same key for messages or associated data that are not equal. Reusing the nonce voids all security claims.

The number of bytes in P and A must be at most $2^{50} - 1$ each.

The output of authenticated encryption is a ciphertext $(C;T)$ obtained by concatenating an unauthenticated ciphertext C and a tag T. The length of C is the same as the number of bytes of P.

The tag length is 16 bytes, but shorter tag lengths can be used by truncating to the desired number of bytes. Tag lengths below 8 bytes are forbidden.

It is assumed that the length of P, A in bits are multiples of 8, i.e., P and A consist of a string of bytes.

The inputs to authenticated decryption are a ciphertext C, an authentication tag T, associated data A, a public message number (nonce) N, and a key K, with the characteristics described above.

The output of authenticated decryption is a failure symbol (eg, $\perp$ in the pseudocode or -1 in the code) if the tag verification fails, or a plaintext P such that the encryption of P with key K, nonce N and associated data A produces (C,T) as output.

1.2 General Structure

Shamash is based on the sponge and duplex constructions ([BDPV, BDPV2]), taking advantage of the recent improvements in security given by ([JLM]).

This type of authenticated ciphers have been extensively studied in the last couple of years, and many proofs of security have been given on the structure itself, so we have taken this approach and concentrated on the security of the underlying permutation.

The internal state is 320 bits. It will be described as composed of five 64-bit words W_0, W_1, W_2, W_3, W_4. After each initialization, absortion of an associated data block, processing of a plaintext or ciphertext block and at finalization, a series of rounds are done over the entire internal state. This rounds are permutations of the state, and they are all the same except for some round constants. The round structure will be described in 1.3. For a pseudocode of the general structure, see Chapter 9.

1.2.1 Initialization

In the initialization step, we load the key into W_0, W_1 and the nonce into W_2, W_3. (note: all loads and stores are little endian). W_4 is filled with a xor of rotations of W_0 and W_1 and a constant:

$$W_4 = Rot(W_0, 32) \oplus Rot(W_1, 32) \oplus cst$$

The constant is 0xff.

After this load, 12 rounds are executed.

1.2.2 Processing the associated data

The associated data is loaded, 128 bits at a time, into the words W_3, W_2. After each load, 9 rounds are executed.

After all the associated data has been processed, the constant 1 is xored into W_1, as a domain-separator between the associated data and the message (plaintext/ciphertext).

1.2.3 Processing the plaintext/ciphertext

The encryption/decryption is a stream cipher using bits of the internal state.

Each block of 128 bits of plaintext is xored with W_3 for the first 64 bits of the block and with W_2 for the second 64 bits, to form the ciphertext. The ciphertext is then loaded into W_3, W_2, in that order. A similar, but inverse, operation is done for deciphering: the ciphertext is xored with W_3, W_2 to obtain the plaintext, and then the ciphertext is loaded into W_3, W_2.

After each such loading of a block, 9 rounds are executed.

1.2.4 Finalization

After the last plaintext/ciphertext block has been produced, the 9 rounds described in 1.2.3 are done anyway, and an additional 3 rounds are also executed, for a total of 12 rounds.

Then the tag is produced from the contents of W_0 and W_1.

1.2.5 Processing incomplete blocks

If the length of associated data or the message is not an exact multiple of 16 bytes, then the following procedure is done with the last ("incomplete") block:

Below when we say "absorbed"we mean xored into the case of the associated data, or the procedure detailed in 1.2.3 in the case of the plaintext/ciphertext.

The incomplete block is absorbed into W_3 and the constant 1 xored after that if the block length is less than 8 bytes, or else the first 8 bytes of the block is absorbed into W_3 and the rest into W_2, with a xor of 1 after that. (note then that if the last block is exactly 8 bytes, they will be absorbed into W_3 and then a 1 will be xored into W_2).

After that, the constant 1 is xored into W_1 as a means of distinguishing the complete versus the incomplete cases.

This is equivalent to think that all the time we are actually absorbing 129 bits instead of 128, by padding each complete block with the bit 0 and an incomplete block with 10...01, where the last bit goes into W_1 rather than W_2, W_3.

1.3 Round Structure

The round consists of three layers:

1. Xor a round constant with W_0.
2. A substitution layer.
3. A diffusion layer.

The round constants are $(11*i)\oplus$ 0xff where i is the round number, except during the processing of the associated data in which they are $(11*i)\oplus$0xad.

1.3.1 Substitution Step

The substitution step consists in the passage of the state through 64 5-bit to 5-bit Sboxes. All 64 Sboxes are in fact the same Sbox. The passage through the Sbox is thought "bitwise"that is the bit 0 of each of the words $W_0, ..., W_4$ are passed through the Sbox, then the bit 1 of each word is passed through the Sbox, etc.

The Sbox is:

$$\begin{aligned} S &= \{16, 14, 13, 2, 11, 17, 21, 30, 7, 24, 18, 28, 26, 1, 12, 6, \\ &\quad 31, 25, 0, 23, 20, 22, 8, 27, 4, 3, 19, 5, 9, 10, 29, 15\} \end{aligned}$$

meaning that $S[0] = 16; S[1] = 14; S[2] = 13; S[3] = 2$ etc.

The order of the words is that W_0 represents the least significant bit of each number between 0 and 31, W_1 the next bit, etc.

The boolean component functions, which are useful for a bitsliced implementation of S, are the following: If we denote a number i between 0 and 31 as $i = 2^4y + 2^3u + 2^2z + 2y + x$ where x, y, z, u, v are bits, and denote $S[i]$ as $2^4 f_4 + 2^3 f_3 + 2^2 f_2 + 2f_1 + f_0$ then the algebraic normal form of these functions is:

$$\begin{aligned} f_4 &= 1 + zv + xv + u + z + xy + y + x \\ f_3 &= uv + xv + v + yu + z + y + x \\ f_2 &= uv + v + zu + u + xz + y + x \\ f_1 &= yv + v + zu + u + yz + z + x \\ f_0 &= v + xu + u + yz + z + xy + y \end{aligned}$$

where $+$ is addition in $GF(2)$.

A more compact form is the following:

$$\begin{aligned} f_4 &= 1 + u + (v \text{ OR } z) + (x \text{ OR } (y + v)) \\ f_3 &= z + (u \text{ OR } y) + (v \text{ OR } (x + u)) \\ f_2 &= y + (z \text{ OR } x) + (u \text{ OR } (v + z)) \\ f_1 &= x + (y \text{ OR } v) + (z \text{ OR } (u + y)) \\ f_0 &= v + (x \text{ OR } u) + (y \text{ OR } (z + x)) \end{aligned}$$

It is the "simplest"Sbox in terms of boolean variables that we could find which is "optimal"with respect to the differential and linear characteristics. (of course there is a whole family of them, permuting the variables and exit functions).

1.3.2 Diffusion Step

The diffusion layer consists of a series of rotations and xors, in three steps.

1. Each 64-bit word is rotated two times and the results xored with the original word. That is each word is changed in the following manner:

   $$w \mapsto w \oplus Rot(w, a) \oplus Rot(w, b)$$

   for some constants a, b, different for each word.

   For any constants $a, b \neq 0$, $a \neq b$ the above transformation is invertible. The inverse is a function of the form $w \mapsto \bigoplus_{i=0}^{t} Rot(w, a_i)$ for some constants and some t.

   The constants a and b for each word have been chosen so that the inverse of each of the five transformations has either t maximum possible (43) or second maximum possible. (37).

   The numbers are, assuming that Rot is rotation to the right, and where t is the number of terms of the inverse:

	a	b	t
W_0 :	43	62	37
W_1 :	21	46	37
W_2 :	58	61	43
W_3 :	57	63	37
W_4 :	3	26	37

2. The previous step only diffuses bits "horizontally" within each word. In this step we mix "vertically" by multiplyng each "column" of the state by a $5x5$ matrix over $GF(2)$ with differential and linear branch number equal to 4. The matrix is:

   $$\begin{array}{ccccc} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{array}$$

   In terms of words, this can be computed as:

   $$\begin{aligned} W_i &= W_i \oplus W_3 \oplus W_4, \quad i = 0, 1, 2 \\ W_i &= W_i \oplus W_0 \oplus W_1 \oplus W_2 \quad i = 3, 4 \end{aligned}$$

3. The previous step diffuses vertically but with the bits "aligned" so in a final step, we de-align the words by rotating each of them (one is left fixed, since relative positions is all that is needed). In this step the rotations are by bytes, to facilitate implementation in 8-bit environments. Specifically:

   W_i is rotated $2i + 1$ bytes to the right, $i = 0, 1, 2, 3$, while W_4 is left fixed.

Chapter 2

Design Rationale

2.1 Design rationale for the choice of the general structure

We thought at first of some block cipher based design, but later preffered a permutation based basic structure based on the sponge and duplex family of structures because this design avoids the complexity of having a separate tag generator and round key generation.

It is a design that has been well studied during the last decade, and has been used extensively, among others by the winner of the SHA-3 competition (for hash) and several candidates of the CAESAR competition, among them one of the recently (a few days ago as of the moment of writing) proclaimed member of the portfolio for lightweight cryptography, Ascon. Although when we choose the general design Ascon had not been yet selected as a member of the portfolio, it was one of the finalists, had been thoroughly studied and the only attacks came from some weakness of the permutation, not the general structure.

2.2 Design rationale for the state size

As for the state size, in Shamash we decided to use a 320-bit size due to the following:

1. Due to the NIST requirements, at least 128+96 bits (key size plus nonce size) are needed. In order to facilitate implementation, we preffered to use a 128 bit nonce. So the minimum size is 256 bits. In another submission we present a design with the 256 bit size, but in this submission we decided on a 320-bit size for the rest of the reasons below.

2. For the rounds used in the permutation, we decided to use the well known standard of mixing with a constant, a substitution step and a diffusion step. For the substitution step, we decided to use a single Sbox used in parallel in a bit-wise manner, method popularized by Serpent and other ciphers, with its advantage in speed and resistance against side channel attacks.

   Although the cipher is oriented to restricted environments, usage in the 64-bit processors should also be reasonable, and working with 64-bit words seemed reasonable.

   That means that if we use a 256-bit state we would have four 64-bit words, and hence, by our desire to use a bit-wise implementation, a 4-bit to 4-bit Sbox. The best differential and linear probabilities of a 4-bit to 4-bit Sbox are 1/4 each. If instead of that we use a 320-bit state, then we can use a 5-bit to 5-bit Sbox. For those Sboxes, the best differential and linear probabilities are now 1/16. This means that we only need 32 active Sboxes throughout the rounds to achieve security. Since in each round there could be up to 64 possible active Sboxes, a well designed diffusion step should take care of that bound. So this is a point in favor of a 320-bit design.

3. In the original duplex construction, given a $(c + r)$-bit state, where data is processed in r-bit chunks, the security is proportional to $2^{c/2}$. This means that if we want 128-bit security we would need a $256 + r$ state. In order to not suffer too much of a speed penalty, $r = 64$ seems reasonable, so under the original security claims of the design, one would need a 320-bit state.

   However, in [JLM] it is proved that in the case of secret key usage, this bound can be reduced. This however only reduces the security of the general construction to the security of the underlying permutation. In the first stages of the design we decided that despite [JLM] we would prefer to be conservative and not allow the attacker access to more than 64 bits of the inner state, and so we further cemented the idea of using a 320-bit state.

   We later changed our minds about the rate r (see 2.3) but we wanted to reuse the design for an unkeyed hash function. In that setting, the $2^{c/2}$ bound apply, so with a 320-bit design we could simply use the same design with a rate $r = 64$ for the hash and with a rate $r = 128$ for the aead.

2.3 Design rationale for the Sbox, the number of rounds and r.

Although originally we chose in part the state size of 320 bits in order to use an Sbox with "optimal" differential and linear characteristics, the selection of the Sbox did not go as we had planned.

It is easy to construct optimal Sboxes, but their boolean components are very complicated, not very suitable for restricted environments. Additionally, in order to facilitate diffusion, originally we wanted to use an Sbox with the property that never a difference of hamming weight one would produce an exit difference of hamming weight one. Unfortunately we were unable to find an optimal Sbox with that characteristic and reasonable boolean functions. (in fact, we didn't find any optimal one with that property, with reasonable or not reasonable booleans).

So we decided to forego that requirement and make the diffusion take care of that. (see 2.4)

That still left the problem of finding any optimal Sbox with reasonable boolean functions, which was further complicated because we wanted each boolean function to depend on all input variables.

Eventually we found one (in fact several) and we took the one which seemed to us to have the most compact boolean functions among the optimal Sboxes, in fact there is only one boolean function $f_0(v, u, z, y, x) = v + (x \text{ OR } u) + (y \text{ OR } (z + x))$ and the others are $f_i = f_0 \circ Rot^i$, where $Rot(v, u, z, y, x) = (x, v, u, z, y)$, except that $f_4 = 1 + f_0 \circ Rot^4$.

In fact our original design was even simpler in the fact that it was of a type $f+ = (b \text{ OR } e) + (c \text{ OR } (e + b))$. However this design had the problem that in it all the differential probabilites $\alpha \mapsto \alpha$ with α of weight one were non zero. In order to make sure that a differential probability of weight one didn't stay in the same word, we changed the design so that, although there are non zero probabilities $\alpha \mapsto \beta$ with both α and β of weight one, it never happens that with $\alpha = \beta$.

Also, if we were to use $f_4 = f_0 \circ Rot^4$ instead of $f_4 = 1 + f_0 \circ Rot^4$, we would have the property $S \circ Rot = Rot \circ S$. Although the diffusion destroys this symmetry because the constants used for the rotations in each word are different, and the round constant also destroys this symmetry, and the initial state is not rotation invariant, and the domain separator is xored into only one word, we added the constant 1 to be on the safe side.

The other problem left was that we could not find a reasonable optimal Sbox with degree greater than 2. (we found several with degree three and four, for instance, but not with simple boolean functions). An Sbox of degree two has the problem that it may allow some algebraic attacks. ([AM, BC, DEMS]).

In order not to suffer too much of a penalty, we started with a working design with 6 rounds of the permutation. That gives a bound of $2^6 = 64$ on the degree of the permutation, way below 320.

Although Ascon, Fides and others also use Sboxes of degree two and has some attacks based on that, the attacks do not carry over the cipher itself. So we could go the same route and "trust" that no attack of this kind would be found against the cipher itself. However, a standard must last many years, and "attacks never get worse". So we considered that given the risk of a future attack of this type, we wanted more protection for these type of attacks.

So, we had a choice:

1. Use a more complicated optimal Sbox

2. Use an Sbox with reasonable functions of degree 3 or 4, not optimal

The first option was not reasonable in the context of lightweight cryptography. As for the second, we did found several candidates with reasonable functions, that had differential probability of 1/8 and linear probability of 9/64 and were of degree 4. Degree 4 in the boolean functions would make the bound 4^6 for the degree useless since $4^6 > 320$ (in fact 4^5 is already greater than 320), and any algebraic attack would have to be constructed in a more difficult way. However, we were not satisfied with the differential and linear probabilities of some differential characteristics and linear trails that we found. Although it seemed that with 4 rounds we already obtained the desired level of security (in the differential case) or close to it (in the linear case), we felt that the margin of security was too close for 6 rounds. Since, despite the recent algebraic attacks on some ciphers, we felt that the differential and linear cryptanalysis attacks were the most dangerous ones, giving rise to possible forgeries or recovery of key bits, we wanted to make sure that these attacks were out of the question.

So we would need to increase the number of rounds.

But in that case we could also increase the number of rounds with the optimal Sbox, to provide better protection against algebraic attacks.

So, we could use 8 rounds with a non optimal Sbox (to provide a comfortable security margin for linear and differential attacks) or 9 for an optimal Sbox to provide protection against algebraic attacks (because $2^9 > 320$ but $2^8 < 320$). (A possibility was to use two Sboxes, one optimal of degree two in some rounds, and another non optimal of degree 4 in other rounds, but this seemed to us unsuitable for light cryptography.)

The 8 round non optimal version would be faster, but ultimately we were more comfortable with the large differential and linear security margin of the 9 round version with the optimal Sbox. Additionally, while a degree 2 Sbox is a weakness against algebraic attacks, it is an asset in regards to threshold implementations and masking, so that was an additional point for the second option.

The speed penalty however was too large so we changed our design to read data in 128-bit blocks instead of 64-bit blocks, using the results of [JLM] and the fact that a 9 round version complicates things for attackers even though they have access to 128 bits of the internal state instead of 64 bits.

2.4 Design Rationale for the diffusion step

For the diffusion step we wanted simple operations, namely xors and rotations.

The Sbox operates "vertically"on the state. So the diffusion needs to spread bits "horizontally", by means of a transformation with branch number at least 3. (so a 1 bit change produces at least 2 bit changes). However, as explained in 2.3, we could not find an Sbox that had zero differential and linear probabilities of the hamming weight one into hamming weight one, so the diffusion has to take care of that too, and have some "vertical"component. (an alternative would be to use a "horizontal only"diffusion and trust that with enough active Sboxes, eventually a weight one difference will produce a weight two difference, but we don't like this approach).

The simplest way of producing diffusion horizontally is to xor to a word a shifted copy of itself, but that leaves some bits that do not "diffuse", hence its branch number is only 2. The next simplest way is to xor to a word a rotated copy of itself. This is not invertible however (the all 1 and all 0 words both go to the all 0 word) so the minimum is to xor to a word two rotated copies of itself, which is invertible. (since the polynomial that represents the transformation is coprime with $1+x$, for more details see [R]). Another possibility is to use $x \mapsto (d\&x) \oplus Rot(x,b)$ for some constants d, b suitable chosen, for example b odd and d not all 0 or all 1.([SV]). This last one is simpler, but it has the same problem as $x \mapsto x \oplus Shift(x,b)$, that it has branch number 2. So we chose $x \mapsto x \oplus Rot(x,a) \oplus Rot(x,b)$ for some constants $a, b \neq 0$, $a \neq b$. The choice of the constants was made regarding the inverse of the operation $x \mapsto x \oplus Rot(x,a) \oplus Rot(x,b)$. This inverse is a function of the form $w \mapsto \bigoplus_{i=0}^{t} Rot(w, a_i)$ for some constants a_i and some t.

The constants a and b for each word have been chosen so that the inverse of each of the five transformations has either t maximum possible (43) or second maximum possible (37). We did not chose in all of them the maximum possible because we saw some similarities in their properties for different constants that achieved this maximum.

For the vertical diffusion an ideal would be to use an MDS matrix (ie, with branch number 6). However, we wanted to maintain the operations simple, so we did not wanted to operate on $GF(2^a)$ for $a > 1$, so we wanted a matrix over $GF(2)$. There are no 5×5 MDS matrices over $GF(2)$, and in fact there are no matrices with branch number 5 (a 5×5 matrix with branch number 5 necessarily would need to have each column with either 4 ones or 5. A 5×5 matrix in which all columns have 4 ones is not invertible, hence we need a column with five ones. The xor of that column with another with 4 ones gives a column of weight 1, hence the branch number of such a matrix is 3). So the best we can hope for is a matrix with branch number 4. We also want one which can be coded efficiently so we settled with:

$$\begin{matrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{matrix}$$

We wanted to have a 0 in position (5,5) to better interact with the final part of the diffusion:

This vertical part spreads the bits vertically, but they remain aligned, so the number of active Sboxes is not changed by this permutation, only the weight of the differences.

Hence we rotate the words as a final step. Since we only want to dealign the words, only 4 rotations are needed, but then one word is left fixed (we chose the last one). To prevent any possible fixed point, we made sure that the entry (5,5) of the vertical diffusion matrix is 0 and all other entries in that row are one.

The linear branch number of the matrix is also 4 (the linear and differential branch number of a MDS matrix are necessarily equal, but this is not true in general. In this case we made sure that is also 4).

2.5 Design rationale for the round constants

We did not want round constants that were too big, but we wanted them to offer enough bits of difference between rounds. The choice $i \oplus const$, where i is the round number, seemed natural but with few bits of change, so we chose $(11 * i) \oplus const$ since this offers more bits of change. For the constant we chose the simple constant $0xff$, except that we wanted to have a domain separator between the associated data and the message, so for the **a**ssociated **d**ata part we used the constant 0xad. Since we also have as domain separator a xor of 1 between the associated data and the message, this part may be unnecessary, but having two different ways of obtaining domain separation seemed a good idea.

Chapter 3

Cryptanalytic Attacks

3.1 Differential and Linear Attacks

The only nonlinear component of Shamash is the Sbox.

The Sbox has been chosen to have the best possible differential and linear probabilities for an Sbox of 5 bits into 5 bits, namely, 2^{-4}.

This means that if there are at least 32 active Sboxes throughout the rounds, we have provable security since $(2^{-4})^{32} = 2^{-128}$

For example, if there is an average of 4 active Sboxes per round during the 9 rounds, any differential characteristic or linear trail will have probability less than or equal to $(2^{-4})^{4\times 9} = 2^{-144}$.

The diffusion however is much better than that:

If we consider only the evolution of a difference through the difusion part (ignoring the Sbox) then we obtain the following results for up to 4 rounds:

For 1,2 and 3 active Sboxes at the start, of any possible weight, the number of active Sboxes at the star of each round for 4 rounds (plus the start of the fifth) that gives a minimum total in each case are:

1 9 39 59 63 for a total of 171 active Sboxes.

2 6 40 60 63 for a total of 171 active Sboxes.

3 9 41 61 62 for a total of 176 active Sboxes.

For 4 Sboxes all of weight 1:

4 6 37 57 63 for a total of 167 active Sboxes.

For 5 Sboxes all of weight 1: 5 9 39 62 62 for a total of 177 active Sboxes.

For the inverse diffusion, if we would want to mount a forward and backward attack, the numbers are:

1 58 63 59 59 for a total of 240 active Sboxes.

2 41 63 62 63 for a total of 231 active Sboxes.

3 37 49 62 59 for a total of 210 active Sboxes.

4 35 48 61 63 for a total of 211 active Sboxes. (restricted to 4 of weight one)

5 22 61 59 62 for a total of 209 active Sboxes. (restricted to 5 of weight one)

Although this is only a partial analysis, it is enough to show security:

The minimum number of active Sboxes in two rounds when there are at most 3 active Sboxes either at the start of the first round or at the start of the second is 8 (2 at the start of the first, 6 at the start of the second). (although the table we gave lists the minimum over 4 rounds plus the start of the fifth, the number of the minimums from one round to the start of the second coincide with the numbers stated above for the cases 1,2,3, and 4 active Sboxes at the input). Hence the other possible case would be 4 on both rounds, any other case would be higher.

So the minimum number of active (differential) Sboxes in two rounds is 8. Hence in 8 rounds the minimum number of active Sboxes is 32 (although the number is likely much higher) and hence we have provable security for a classical differential cryptanalytic attack.

For linear cryptanalysis, the corresponding minimal number from one round to the next is not 8 but 6, but when this happens there are 12 active linear Sboxes in the next round.

By using a differente search that is slower and takes into account the passage through the Sbox, the best numbers (from the point of view of the attacker) that we could find for 3 rounds plus the start of the fourth was 73 linear active Sboxes with a total linear probability of 2^{-292}. (round by round: 48 in the first round,17 in the second, 2 in the third,6 at the start of the 4th)

3.2 Differential Factors and Linear Structures Attack

Let S be an Sbox. Then S has a differential factor λ for the output difference μ if given μ, λ it happens that for all x, y with $S(x) \oplus S(y) = \mu$, we also have $S(x \oplus \lambda) \oplus S(y \oplus \lambda) = \mu$.([TO])

Sboxes with differential factors can allow some differential-linear attacks ([TO]). The Sbox of Shamash has no differential factors so these kind of attacks cannot proceed.

Definition: ([E]) An $n \times m$ S-Box S is said to have a linear structure if there exists a nonzero vector $a \in GF(2)^n$ together with a nonzero vector $b \in GF(2)^m$ such that $b \odot S(x) \oplus b \odot S(x \oplus a)$ takes the same value $c \in GF(2)$ for all $x \in GF(2)^n$.

Linear structures can also be used in some kind of linear-differential attacks. While the Sbox of Shamash has 31 linear structures, the inverse of the Sbox has none, which limits this kind of attacks. For comparison, the Sbox of Ascon has 91 linear structures and its inverse has 2 of a particular kind called "undisturbed bits"([T]) which allow some attacks on 5 of the 12 initialization rounds of Ascon, although this attack does not carry over the scheme. Since the inverse of the Sbox of Shamash has none such structure, we think that any such attack in the case of Shamash will be very difficult and would also not carry over the whole scheme.

3.3 Algebraic Attacks

Algebraic attacks like zero-sum and cube attacks are a concern for Shamash given that the boolean components of the Sbox are all of degree 2. Attacks of this type have been mounted against the permutations of Ascon and other algorithms.([AM, BC, DEMS])

Although in most cases those attacks on the permutations do not carry over an attack on the general structure itself, we have chosen 9 rounds so that the "easy" bound on the degree of the permutation given by $2^{\text{number of rounds}}$ ceases to be meaningful (since $2^9 > 320$) and because of the strong diffusion, to have a high algebraic degree of the permutation.

We expect that a zero-sum attack against Shamash will be very difficult to mount, and in light of the results obtained for Ascon and other ciphers, we expect that even if such an attack is mounted against the indifferentiability of the permutation, the attack will not carry over any attack against the security of the cipher itself.

3.4 Resistance against related keys

If the key is not mixed well, some attacks use keys that are related to improve the differential probabilities of an attack. (eg, [BK] exploit some weakness in the key expansion of AES). In our case the key is passed through 12 rounds of the permutation before interacting with any data, and thus a related key attack is unlikely.

3.5 Resistance against related nonces

This is similar to the previous point except that the nonces may more easily be under the control of the attackers. But, as before, the nonce is mixed with the secret key during 12 rounds of initialization, so this kind of attack is unfeasible.

3.6 Integral Cryptanalysis

Integral Cryptanalysis ([KW]) is more suited to a structure like AES, which moves the bits of the Sbox in blocks, and then combines them linearly. The structure of Shamash separated the bits of the Sbox during the diffusion into separated parts of the state so this type of attack seem unlikely.

Chapter 4

Advantages of Shamash

Shamash has the following advantages:

1. It uses a well studied design that has been used many times before.
2. It uses an Sbox that is optimal against differential cryptanalysis and linear cryptanalysis.
3. The Sbox has a simple boolean representation.
4. Each output bit of the Sbox depends on every input bit, and it depends nonlinearly on four of the five.
5. The differential probabilities of the Sbox together with the diffusion and the number of rounds give an easy proof of the security against classical differential cryptanalisis. The bound we obtained is very conservative and it is likely that the cost of any differential attack is well above the security level.
6. Although the algebraic degree of the Sbox is only 2, the number of rounds is sufficiently high as to make any algebraic attack non-trivial.
7. The fact that the degree of the Sbox is two, while making it more vulnerable to algebraic attacks, make it better for threshold implementations and masking.
8. All operations are simple operations both in hardware and software: xors, ors, rotations.
9. It can easily be implemented in such a way that the same code works on little endian and big endian machines.

10. The size of the state is near the minimum possible given the NIST requirements.

11. Although not nearly as fast as AES-OCB for example, its speed in 64-bit machines is reasonable.

12. The structure of the round allows for faster implementations in FPGA at the cost of gate counts, or the permutation step can be implemented to work one Sbox per clock at the cost of speed for gains in gate count, or something in between.

13. The inverse permutation is not needed, so there is no need for different circuitry for the inverse cipher, only the order in which the xor between the state and the message is done when the message is plaintext or ciphertext.

14. If there is no associated data to be authenticated, Shamash starts processing the message part directly, with only a cost of a xor of 1.

15. The padding is done in such a way that a complete block of data does not require extra processing compared to incomplete blocks.

16. The key can be changed with every message if that is wanted, at no extra cost.

17. The state at the time right before the first process of a data block depends bijectively on the pair (key,nonce).

However, Shamash has the following disadvantages:

1. It processes data blocks in a serial manner, so it cannot take advantage of processors that would allow to process data blocks in parallel, something that AES-OCB for example is able to do. However, given that it is oriented towards light weight cryptography, we do not feel that this is too much of a disadvantage.

2. It is not "the fastest cipher in town", although we think that its speed is acceptable: the reference version can run at 23 cycles per byte, and the (not very) optimized version at 16 cycles per byte.

3. The algebraic degree of the Sbox is only 2. This is shared by many other designs, and in Shamash case care has been taken to lessen the impact of the low degree of the Sbox. As stated above, this low degree has also advantages.

4. It does not allow repetition of the nonce.

5. It does not have any insurance against recovery of the inner state: because of the last item of the list of advantages, recovery of the inner state allows recovery of the key. However we do not expect any attack that can recover the inner state if the nonce is not repeated and no plaintext is returned if the verification of the tag fails.

Chapter 5

The name Shamash

The name Shamash means "**Sh**ort **A**uthenticated **M**essage **A**nd **S**ecure **H**ash" and basically we invented those words in order to have the name, since Shamash is also the akkadian name of the god of the sun, who had the power of light over darkness and evil, and thus was also the god of justice. (For example, according to legend, Hammurabi received its famous tablet from Shamash .Also Shamash helped Gilgamesh in his quest). As god of justice he detected falsehoods and lies and was the god of truth, hence its name seems adequate for an authenticated cipher.

Chapter 6

Security Goals

goal	bits of security
Confidentiality for the plaintext	128
Integrity for the plaintext and ciphertext	128
Integrity for the associated data	128
Integrity for the public message number	128

That is, we expect that any attack on the confidentiality of the plaintext will need 2^{128} effort and a forgery of the plaintext, associated data or public message number cannot be made with probability greater than 2^{-128}. (i.e., an expected 2^{128} attempts need to be made before a forgery is accepted as valid).

This assumes that P is at most 2^{50} bytes long and A is at most 2^{50} bytes long and that the public message number is not repeated within the lifetime of a key, i.e., it must be a nonce. Repetition of a nonce voids all security goals.

Note: section 3.1 of the NIST call specify that the plaintext input is a byte-string, so we do not distinguish between a plaintext P_1 of p bytes, considered as $8p$ bits, with the last byte with r trailing 0s, and another plaintext P_2 of length $8p - r$ bits with the same initial $p - 1$ bytes as P_1 and the remaining $8 - r$ bits the same as the initial $8 - r$ bits of the last byte of P_1.

Shamash is specified with a 128 bit tag. If an implementer wishes to change this by truncating the tag, then if the tag is truncated to t bits, the numbers 128 in the table above in the last three entries change to t. However, this is assuming that the implementer will implement a single tag length. If an implementer implements Shamash by allowing multiple tag lengths, then the numbers for the integrity of the plaintext,ciphertext, associated data and public message number collapse, for all tag lengths, to the lowest tag length implemented, unless the implementer ensures that the same pair (key,nonce) will not be used for different key lengths. We strongly discourage implementing multiple tag lengths.

Also, note that if an implementer implements both Shamash and Shamashash, with the second using the first as a black box, then the tag for Shamash cannot be truncated. (see chapter 7).

Chapter 7

Shamashash v1

7.1 Specification

7.1.1 Introduction

Shamashash is a 256-bit hash function related to the Shamash authenticated encryption scheme. Both the input and the output are byte strings, that is, it is assumed that the lengths of the message and the hash are both multiples of 8 bits.

The algorithm is permutation based, and it is an instantiation of the AEAD algorithm Shamash, described earlier in this document.

Shamashash uses the encryption part of Shamash as a black box (there is no need for the decryption part). The way it uses it is to transform the AEAD Shamash into a sponge ([BDPV])

Shamash, as an AEAD, receives a key of 128 bits, a nonce of 128 bits, some associated data and plaintext and outputs a ciphertext, of which the last 128 bits are considered the authentication tag. It works by absorbing the associated data and plaintext in 128-bits blocks.

Shamashash receives some byte string and outputs a 32-byte string, which is the 256-bit hash of the input.

It transforms the input string in inputs for Shamash in the following way:

1. The first 8 bytes of the input are used to construct the first 8 bytes of the nonce for Shamash. If there are less than 8 bytes of input, then only those bytes are used and the rest of the first 8 bytes of the nonce are set to 0.

2. The 9th byte of the nonce (of index 8 if we start counting at 0) is set to 9 if there are more than 8 bytes of input, or the number of bytes of the input if there are 8 or less bytes of input.

3. The other 7 bytes of the nonce are set to 0.
4. The key is set to 0.
5. The remaining bytes of the input, if any, are used to construct the associated data, by constructing an associated data twice the length of the remaining bytes and placing the bytes of the inputs in the even positions (counting from 0) and letting the odd bytes of the associated data be 0.
6. The plaintext is a 48-byte string of 0s.

Shamash will output a ciphertext of length 48+16=64 bytes. The even bytes of that ciphertext are used to construct the hash Shamashash .

7.2 Security Claims

Shamash absorbs data in 128-bit blocks, but here we format the input to Shamash in such a way that half those bytes are 0 for the nonce and the associated data, and all 0 for the plaintext. So in effect this version absorbs the input at a rate of 64 bits. Since Shamash has an inner state of 320 bits, this leaves a capacity of 256 bits, and hence by the $c/2$ security bound of the sponge construction we should have 128 bit security. Actually there is a theoretical slight loss of security since we also use 4 bits extra bits in the construction of the nonce and two extra bit of padding for inputs whose lengths are not multiple of 8 bytes, because of the structure of Shamash, so we claim 126 bit security.

In accordance to the NIST call, we do not distinguish between messages one of which is a truncation of the other by a number of bits less than 8.

7.3 Security Analysis

Since Shamashash uses Shamash as a black box, all the security analysis for Shamash carry over Shamashash , except that since Shamashash , as a hash, does not use a key, we absorb data at half the rate of Shamash.

7.3.1 Advantages and disadvantages

It has the same advantages and disadvantages of Shamash, with an additional disadvantage: since it has half the rate, it is slower.

One advantage in the context of light weight cryptography is that one needs minimal extra circuitry for the hash function Shamashash since it uses Shamash as a black box.

Chapter 8

Intellectual Property

We, Daniel Penazzi and Miguel Montes, do hereby declare that the cryptosystems, reference implementations, or optimized implementations that we have submitted, known as Shamash and Shamashash, is our own original work.

We further declare that we do not hold and do not intend to hold any patent or patent application with a claim which may cover the cryptosystems, reference implementation, or optimized implementations that we have submitted, known as Shamash and Shamashash.

Bibliography

[AM] J.-P. Aumasson and W. Meier. Zero-sum distinguishers for reduced Keccak-f and for the core functions of Luffa and Hamsi. presented at the rump session of Cryptographic Hardware and Embedded Systems - CHES 2009, 2009.

[BDPV] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche On the security of the keyed sponge construction

[BDPV2] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche Duplexing the sponge: single-pass authenticated encryption and other applications

[BK] Alex Biryukov and Dmitry Khovratovich, *"Related-key Cryptanalysis of the Full AES-192 and AES-256"*, Advances in Cryptology-ASIACRYPT 2009 Lecture Notes in Computer Science Volume 5912, 2009, pp 1-18.

[BS] Biham, E. and Shamir, A. (1991). Differential cryptanalysis of DES-like cryptosystems. J. Cryptology, 4(1):3-72.

[BC] Christina Boura, Anne Canteaut. A Zero-Sum property for the Keccak-f Permutation with 18 Rounds

[DEMS] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schlä ffer. Cryptanalysis of Ascon.

[E] Evertse, J.-H. (1987). Linear Structures in Blockciphers. In Chaum, D. and Price, W. L., editors, EUROCRYPT, volume 304 of Lecture Notes in Computer Science, pages 249266. Springer.

[JLM] Jovanovic, P., Luykx, A., and Mennink, B. (2014). Beyond 2 c/2 security in sponge-based authenticated encryption modes. In Sarkar, P. and Iwata, T., editors, Advances in Cryptology - ASIACRYPT

2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, PartI, volume 8873 of Lecture Notes in Computer Science, pages 85104. Springer.

[KW] L.R. Knudsen and D. Wagner. Integral cryptanalysis. In Fast Software Encryption - FSE 2002, volume 2365 of Lecture Notes in Computer Science, pages 112-127. Springer-Verlag, 2002.

[R] Rivest, R. L. (2011). The invertibility of the XOR of rotations of a binary word. Int. J. Comput. Math., 88(2):281284.

[T] Cihangir Tezcan. Truncated, Impossible, and Improbable Differential Analysis of Ascon. Proceedings of the 2nd International Conference on Information Systems Security and Privacy - Volume 1: ICISSP, 325-332, 2016, Rome, Italy

[TO] Tezcan, C. and Özbudak, F. (2014). Differential factors: Improved attacks on SERPENT. In (Eisenbarth and Özturk, 2015), pages 69-84.

[SV] Claus P. Schnorr and Serge Vaudenay. Black box cryptanalysis of hash networks based on multipermutations. In Proceedings EUROCRYPT 94, volume 950 of Lecture Notes in Computer Science, pages 4757. Springer, 1995.

Chapter 9

Appendix: Pseudo code of Shamash

```
AEAD-ENCRYPT($K, N, W, A, P$)
1  INITIALIZE($K, N, W$)
2  PROCESS AD($W, A$)
3  PROCESS PLAINTEXT($W, P$)
4  Do  3 Rounds. // Another 9 rounds have been done at the last part
5        // of PROCESS PLAINTEXT($W, P$) so the total is 12 rounds.
6  Output $W_0$,$W_1$ as tag.
```

Figure 9.1: AEAD-encrypt procedure

ROTR is rotation of a 64 bit word to the right. $\oplus$ is the bitwise xor.

```
AEAD-DECRYPT($K, N, W, A, C, T$)
1  INITIALIZE($K, N, W$)
2  PROCESS AD($W, A$)
3  PROCESS CIPHERTEXT($W, C$)
4  Do  3 Rounds. // Another 9 rounds have been done at the last part
5        // of PROCESS CIPHERTEXT($W, C$) so the total is 12 rounds.
6  if ($T_i$ is equal to $W_i$,$i = 0, 1$)
7        return the generated plaintext.
8  else
9        return $\perp$
```

Figure 9.2: AEAD-decrypt procedure

```
INITIALIZE($K, N, W$)
1  // Initialization
2  $W_i = K_i$, $i = 0, 1$
3  $W_2 = N_0$
4  $W_3 = N_1$
5  $W_4 = \mathrm{ROTR}(W_0 \oplus W_1, 32) \oplus \mathrm{0xff}$
6  Do  12 Rounds.
```

Figure 9.3: Init procedure

```
PROCESS AD($W, A$)
while there is at least 16 bytes of $A$ left to process
    Xor 16 bytes of $A$ into $W_3, W_2$.
    Do 9 Rounds.
Endwhile
if there is between 1 and 7 bytes of $A$ left to process.
    Xor those bytes into $W_3$.
    Xor a 1 to $W_3$ after those bytes.
    Xor a 1 to $W_1$.
    Do 9 Rounds.
Endif
if there are exactly 8 bytes of $A$ left to process.
    Xor those bytes into $W_3$.
    Xor a 1 to $W_2$.
    Xor a 1 to $W_1$.
    Do 9 Rounds.
Endif
if there is between 9 and 15 bytes of $A$ left to process.
    Xor the first 8 of those bytes into $W_3$.
    Xor the rest of the bytes into $W_2$.
    Xor a 1 to $W_2$ after those bytes.
    Xor a 1 to $W_1$.
    Do 9 Rounds.
Endif
Xor a 1 to $W_1$ (delimiter ad/message)
```

Figure 9.4: Process AD procedure

```
PROCESS PLAINTEXT($W, P$)
while there is at least 16 bytes of $P$ left to process
    Xor $W_3, W_2$ with 16 bytes of $P$ to produce ciphertext.
    Load the resulting 16 bytes of ciphertext into $W_3, W_2$.
    Do 9 Rounds.
Endwhile
if there is $t$ bytes of $P$ left to process with $t$ between 1 and 7.
    Xor $t$ bytes of $W_3$ with $t$ bytes of $P$ to produce ciphertext.
    Load those ciphertext bytes into the corresponding bytes of $W_3$.
    Xor a 1 to $W_3$ after those bytes.
    Xor a 1 to $W_1$.
    Do 9 Rounds.
Endif
if there are exactly 8 bytes of $P$ left to process.
    Xor $W_3$ with 8 bytes of $P$ to produce ciphertext.
    Load those ciphertext bytes into $W_3$.
    Xor a 1 to $W_2$.
    Xor a 1 to $W_1$.
    Do 9 Rounds.
Endif
if there is $t$ bytes of $P$ left to process with $t$ between 9 and 15.
    Xor $W_3$ with the first 8 bytes remaining of $P$ to produce ciphertext.
    Xor $t-8$ bytes of $W_2$ with the final $t-8$ bytes of $P$ to produce ciphertext.
    Load the first 8 of the ciphertext bytes into $W_3$.
    Load the rest of the ciphertext bytes into $W_2$.
    Xor a 1 to $W_2$ after those bytes.
    Xor a 1 to $W_1$.
    Do 9 Rounds.
Endif
```

Figure 9.5: Process Plaintext procedure

```
PROCESS CIPHERTEXT($W, C$)
1  while there is at least 16 bytes of $C$ left to process
2      Xor $W_3, W_2$ with 16 bytes of $C$ to produce plaintext.
3      Load the 16 bytes of ciphertext into $W_3, W_2$.
4      Do 9 Rounds.
5  Endwhile
6  if there is $t$ bytes of $C$ left to process with $t$ between 1 and 7.
7      Xor $t$ bytes of $W_3$ with $t$ bytes of $C$ to produce plaintext.
8      Load the $t$ ciphertext bytes into the corresponding bytes of $W_3$.
9      Xor a 1 to $W_3$ after those bytes.
10     Xor a 1 to $W_1$.
11     Do 9 Rounds.
12 Endif
13 if there are exactly 8 bytes of $C$ left to process.
14     Xor $W_3$ with 8 bytes of $C$ to produce plaintext.
15     Load the ciphertext bytes into $W_3$.
16     Xor a 1 to $W_2$.
17     Xor a 1 to $W_1$.
18     Do 9 Rounds.
19 Endif
20 if there is $t$ bytes of $C$ left to process with $t$ between 9 and 15.
21     Xor $W_3$ with the first 8 bytes remaining of $C$ to produce plaintext.
22     Xor $t - 8$ bytes of $W_2$ with the final $t - 8$ bytes of $C$ to produce ciphertext.
23     Load the first 8 of the ciphertext bytes into $W_3$.
24     Load the rest of the ciphertext bytes into $W_2$.
25     Xor a 1 to $W_2$ after those bytes.
26     Xor a 1 to $W_1$.
27     Do 9 Rounds.
28 Endif
```

Figure 9.6: Process Ciphertext procedure