Lightweight Cryptography on ARM
A presentation proposal

Rafael J. Cruz¹, Tiago B. Reis¹, Diego F. Aranha¹, Julio López¹, Harsh Kupwade Patil²

¹ Institute of Computing – University of Campinas
² LG Mobile Research

Abstract. We present multiple contributions to the efficient software implementation of cryptographic algorithms for ARM devices. The talk summarizes three contributions: (i) LS-designs (represented by Fantomas), their efficient implementation and side-channel security; (ii) techniques to optimize implementations of the PRESENT block cipher, using bitslicing and permutation decomposition. (iii) miscellaneous observations about efficient implementation of dedicated MAC algorithms, such as Chaskey and SipHash. The implementations target the Cortex-M and Cortex-A families of ARM processors. These devices are located towards the mid to lower-end of the spectrum of ARM architectures, and are typical of scenarios considered for lightweight cryptography, such as the Internet of Things. We improve on the state-of-the-art implementations of these algorithms substantially, both in terms of efficiency, security or compactness, by making use of novel algorithmic techniques and features specific of the target platforms.

1 Introduction

The emergence of the Internet of Things (IoT) immediately raises concerns about the security of communications between IoT devices and even of the devices themselves. The fact that an extraordinary number of wirelessly networked devices will continuously store and exchange sensitive data has exposed a larger attack surface (ranging from physical exposure and ease of access to remote availability) and made practical several attack scenarios that were only considered in the research literature.

While designing and developing secure efficient implementations of cryptography is not a new problem and has been an active area of research since the birth of public-key cryptography, the emergence of IoT brings new challenges to this paradigm. In particular, special attention must be given to side-channel attacks, in which operational aspects of the implementation of a cryptographic algorithm may leak internal state information and allow an attacker to retrieve cryptography keys by only observing leakage through the communication channel, power consumption, execution time or radiation measurements. Information leaked through cache latency [1,2] or execution time [3] already allows powerful timing attacks against naive implementations of symmetric and public-key cryptography, respectively. More intrusive attacks also attempt to inject faults at precise execution times, in hope of corrupting execution state to reveal secret information. Consequently, securely implementing cryptography in typical resource-constrained IoT devices is a challenging research problem for the next few years. Optimizing such implementations to strike an ideal balance between resource efficiency and side-channel resistance further complicate matters, beckoning both algorithmic advances and novel implementation strategies.

In this proposal, we present a series of contributions to the efficient software implementation of strong contenders in the lightweight cryptography space:

- Efficient, portable, secure and compact implementation of LS-Designs, instantiated through the Fantomas block cipher. We also study the side-channel security of LS-Designs, by proposing a simple cache-timing attack on the linear diffusion layer; and constant-time, masked versions of the implementation as possible countermeasures. In particular, our unprotected
32-bit implementation achieves speedups from 35% to 66% in the ARM Cortex-M architecture, while consuming considerably less code size. The vectorized implementation improves performance over the state of the art by 40% in the ARM Cortex-A15.

- A technique to optimize the PRESENT block cipher by exploiting the cipher structure to decompose the permutation \( P \) into simpler ones and combine permutations together with the key schedule. This removes the need of applying the permutations at every round of the cipher. A careful implementation of the block cipher under the CTR mode of operation provides up to an 8-factor speedup over state-of-the-art related work.

- Miscellaneous results on the implementation of ARX-based MAC algorithms, such as Chaskey and SipHash. Fully unrolled implementations of the round function and conversion of internal state into registers allow savings in the number of memory operations and provide up to 40% performance improvement in the Cortex-M, while keeping code size controlled.

ARM was selected as the platform of choice, due to its cost-effectiveness, wide availability and stringent resource consumption in terms of power. We further target the Cortex-M and Cortex-A family of processors in the ARM spectrum, because they are representative of different segments of architectures, but still include powerful resources such as DSP-style instructions and cache memory (even in some M4 cores). We now focus on the two main contributions regarding block ciphers, both of which are in the process of submission to academic venues.

## 2 Fantomas

LS-Designs [4] were conceived to address side-channel threats, by combining the advantages of bitslicing-capable ciphers with easy support to regular and masked software implementations. Algorithm 1 presents a generic specification for an LS-Design, illustrating its simplicity and regularity. Instances of a LS-Design cipher are characterized by the choice of bitsliced S-boxes \( S \), an L-box matrix \( L \) acting as the diffusion layer, a number of rounds \( N_r \) and round constants \( C(r) \). In the original LS-Design paper, two ciphers were instantiated and analyzed: Robin, a faster involutive instance that later succumbed to invariant subspace attacks [5]; and the non-involutive candidate Fantomas.

**Algorithm 1** LS-Design construction encrypting plaintext block \( B \) into ciphertext block \( C \) with key \( K \).

1. \( C \leftarrow B \oplus K \quad \triangleright \ C \) represents an \( s \times l \)-bit matrix
2. for \( 0 \leq r < N_r \) do
3.     for \( 0 \leq i < l \) do
4.         \( C[i, *] = S[C[i, *]] \) \quad \triangleright \ S-box layer
5.     end for
6. for \( 0 \leq j < s \) do
7.     \( C[* ,j] = L[C[* ,j]] \) \quad \triangleright \ L-box layer
8. end for
9. \( C \leftarrow C \oplus K \oplus C(r) \) \quad \triangleright \ Key and round constant addition
10. end for
11. return \( C \)

Fantomas employs the 3/5-bit S-boxes from the 3-round MISTY cipher [6]. An important consideration taken by the original authors of the cipher is the number of AND operations in the choice of S-boxes. For security of the masking countermeasure, a lower bound on the number of ANDs is the size of the S-boxes. Because Fantomas employs S-boxes of 8-bit granularity, the S-boxes must contain at least 8 AND operations to be appropriate for masking. There is some
security margin in this design decision because Fantomas employs 11 AND operations between elements of the cipher state. The L-box is presented in Figure 1 and its computation can be seen as a vector-matrix product in $F_2$, as illustrated in the picture.

![Figure 1. Linear layer of Fantomas. The L-box matrix has gray cells for 1 bits and white cells for 0 bits.](image)

We have performed multiple implementations of the cipher: a portable implementation for 32-bit and 64-bit processors, and vectorized code for NEON instructions. The portable implementation employs a simple technique to represent the internal state and simultaneously allow operations over 16-bit or 32-bit data inside the S-boxes. The diffusion layer is performance-critical and presents more obstacles to side-channel resistance, since it is implemented through table lookups on the L-box, thus both protected and unprotected versions of the L-box were implemented. Notice that an attacker able to monitor during encryption what positions of the L-box are used through cache timing information is automatically able to compute the key $K$ by simply XORing internal state before the last key addition together with the ciphertext. This is true even for masked implementations. The unprotected version employs two 256-position half-word precomputed tables, while the constant-time version implements the operation on-line by performing the vector-matrix binary multiplication, where two or four 16-bit values are processed at the same time. In terms of masking, we implemented a standard technique and devised a simple way to mask the key $K$ which forces an attacker to recover all shares in order to mount a cache-timing attack. We argue that either a fully constant-time implementation or the key masking technique are required to properly protect against timing attacks.

There are two main related works that established the previous state of the art for implementations of Fantomas. The most recent is the massive implementation effort from the FELICS framework [7]. We target the same Cortex-M3 processor considered in their work (Arduino Due) and two scenarios are taken into consideration. Scenario 1 considers consecutive encryption and decryption of 128 bytes in CBC mode. In the paper, the best implementation according to their Figure of Merit (FOM) takes 70,197 cycles using 4620 bytes of ROM. The website has more recent numbers for an implementation capable of encrypting and decrypting in 94,921 cycles which consumes 2916 bytes of ROM. Our implementation is 35.4% and 52.2% faster than their implementations, respectively, and competitive in terms of code size with the more compact implementation. In Scenario 2, FELICS reports a range of figures for unprotected Fantomas when encrypting 128 bits in CTR mode, ranging from compact implementation to best execution time. The most compact takes 8335 cycles and 1384 bytes of ROM (520.94 CPB), the most efficient takes 3522 cycles and 2088 bytes of ROM (220.13 CPB) and a good trade-off is found at 4550 cycles and 2184 bytes of code size (284.38 CPB). After the proper conver-
visions, our implementation improves these figures by 66.5%, 20.9% and 37.7%, respectively, by spending only 1916 bytes of ROM. The NEON implementation organizes data in vector registers differently than related work and computes 16 encryptions simultaneously in CTR mode. With help of work [8] from the Fantomas designers, we adjusted timings for vectorized code taking into account platform and benchmarking differences. We then observed an approximate performance gain of 40% of our implementation when compared to the adjusted timings in the Cortex-A15 platform. The protected implementations still introduce a substantial performance penalty, requiring further research work.

3 PRESENT

PRESENT [9] is a lightweight block cipher optimized for hardware implementation whose design was published in 2007. The PRESENT encryption routine receives as input a key $K$, which may consist of 80 or 128 bits, and a 64-bit block of plaintext. The key is processed through a schedule that produces thirty-two 64-bit subkeys $subkey_i$. Then, the plaintext block is processed as usual in a substitution-permutation network: in each one of 31 rounds, the block is XORed with a subkey, every group of 4 consecutive bits is substituted through an S-box $S$ and the bits are repositioned by the permutation $P$, described by the rule given in Equation 1. At the end of the algorithm, the state is once again XORed with a subkey. Concerning the permutation $P$, it holds the interesting property that $P^2 = P^{-1}$ and $P^2$ can be decomposed into two permutations $P_0$ and $P_1$, which we will use later. The encryption function is described in Algorithm 2.

$$P(i) = \begin{cases} 16i \mod 63 & \text{if } i \neq 63 \\ 63 & \text{if } i = 63 \end{cases} \quad (1)$$

Algorithm 2 Encryption in PRESENT of plaintext block $B$ to ciphertext block $C$.

```
1: $C \leftarrow B$
2: for $i = 1$ to 31 do
3: \hspace{1em} $C \leftarrow C \oplus subkey_i$
4: \hspace{1em} $C \leftarrow S(C)$
5: \hspace{1em} $C \leftarrow P(C)$
6: end for
7: $C \leftarrow P \oplus subkey_{32}$
8: return $C$
```

One of our primary concerns is to avoid implementing the S-box as a lookup table, not only to avoid costly memory access operations but also to mitigate possible timing attacks. To this end, we simulate the application of the S-box via bitwise operations, as proposed in [10]. However, using four registers to operate and produce a single bit of output is highly inefficient and, thus, we calculate 16 S-boxes simulations in parallel, reorganizing the cipher internal state.

Now, to further enhance performance, we try to avoid applying the permutation $P$ directly onto the state. To adapt the subkey addition, we need to apply the permutation over the subkeys, so that the correct bits are summed up. Adapting the S-box step is more complicated, because at each round the bits over which we have to operate occupy a specific position in the state, but the property that $P^2 = P_1 \circ P_0$ allows us to prepare different arrangements for the bits before the S-box step, and then repeat them. One of these arrangements consists in no transformation, because the bits are already properly aligned. The other two are in fact a permutation of the bits, but simpler than the actual permutation $P$, since we do not have the strict constraints of which position each bit have to go to, we just need to have the proper bits aligned.
One final optimization that may be carried out is that our proposed organization for one block state uses 16 bits of 4 registers. Since our target architecture has a 32-bit word length, we may simply couple another block of message to be processed simultaneously, assuming the use of a mode of operation that does not cause dependences between different blocks, as is the case of CTR mode.

After implementing all of these ideas, we compared our results with those presented by [7] in a scenario that assumes the key schedule already computed and the subkeys ready to be used in the memory of the device. In this case, to encrypt 128 bits of data using the CTR mode, our implementation takes 2050 clock cycles whereas [7] performs the same work in around 17000 Cortex-M3 cycles. As a result, PRESENT is again competitive with more recent block cipher designs.

Conclusion and acknowledgements

In this work, we have presented performance and security improvements for implementation of block ciphers Fantomas and PRESENT. Together with improved implementation strategies for dedicated MAC algorithms, we describe optimizations targeting the main primitives considered by NIST in its potential standardization effort. Finally, the authors fully acknowledge support from LG Electronics Inc. during the development of the research presented here.

References