The Software/Hardware Co-Design and Implementation of SM2/3/4 Encryption/Decryption and Digital Signature System

Xin Zheng, Chongyao Xu, Xianghong Hu, Yun Zhang, and Xiaoming Xiong

***Abstract*—The security of smart devices is facing great challenges. This article presents a new hybrid cipher framework suitable for such devices. Using software/hardware (SW/HW) co-design method, an efficient encryption/decryption and digital signature scheme based on SM2, SM3, and SM4 algorithms is implemented. First, the framework is partitioned into software and hardware parts based on the analysis result of a pure software solution and the hardware overhead under the conditions of design constraints. Second, the flow of the whole scheme and embedded algorithms are realized by SW/HW co-design. Finally, an improved implementation of SM2/3/4 algorithms is proposed to achieve higher efficiency. In this implementation, some SW/HW modules are parallelized to reduce the running time and enhance the performance. The proposed design is safe and can resist simple power analysis (SPA) attacks. Especially, the AHB bus interface IP and software scheduling approach are adopted in data transfer and manipulation. The design is taped-out on a silicon chip with SMIC 110-nm technology process. The chip uses about 199K logic gates and 1-mm^2 areas. The operating frequency of the design is 36 MHz and the chip consumes 23-mW power. The comparison with similar previous works shows our proposed design is more efficient with speed increasing of more than 10%.**

***Index Terms*—Encryption/decryption, parallel processing, signature/verification, SM2/3/4, software/hardware (SW/HW) co-design.**

I. Introduction

CRYPTOGRAPHY techniques have been recognized as an important aspect in the field of information security [1]. There are three types of cryptographic algorithms: 1) symmetric algorithms; 2) asymmetric cryptographic algorithms; and 3) hash algorithm. The related typical algorithms are AES [2], ECC [3], and SHA-256 [4], respectively. The SM2 digital signature algorithm, SM3 hash algorithm, and SM4 block cipher algorithm are published by Security Commercial Code Administration Office of China. The SM2/3/4 algorithms are also recognized by the ISO/IEC international standard [5]–[7] and will gradually be widely used.

TABLE I
Comparison of the Characteristics of Crypto Algorithms

characteristic	**SM2**	**ECDSA(ECC)**
Structure	elliptic curve	elliptic curve
Computational complexity	full exponential	full exponential
elliptic curve parameters	generated by algorithm	NIST recommend
preprocess	yes	no
hash algorithm	SM3	MD5/SHA-256/SHA-512
Security	discrete logarithm	discrete logarithm
characteristic	**SM4**	**AES**
Structure	unbalanced Feistel	Substitution-Permutation
Calculated rounds	10/12/14	32
Grouping length	128	128
Key length	128/192/256	128
Security	S-box Nonlinearity	S-box Nonlinearity

The SM2 digital signature algorithm is an ECC-based cipher algorithm. The security of SM2 has been proved [8] and it is more secure against the generalized key substitution attacks [9]. SM3 has similar basic structure as SHA-256. With some new techniques added in SM3, such as 16-step all-XOR operation [10], it is more effective against boomerang attacks. SM4 is a block cipher algorithm similar to AES-128 with a simplified round-key generation [11] and it is more effective in defending key-leakage type Trojan [12]. Overall, SM2/3/4 are improved cipher algorithms based on ECC/SHA-256/AES with better efficiency and higher security measurements. For the digital signature algorithm, we compare the SM2 with the latest ECDSA algorithm [13] which is using ECC to simulate the digital signature algorithm. For symmetric encryption algorithms, we compare SM4 with the latest AES algorithms. The comparison results are shown in Table I.

As reported, cryptographic algorithms are previously implemented in pure software or pure hardware-based ASIC [14]. Relatively speaking, pure software encryption requires more system resources and is relatively easy to be attacked.

Manuscript received February 26, 2019; revised May 17, 2019; accepted August 15, 2019. Date of publication September 4, 2019; date of current version September 18, 2020. This work was supported in part by the Science and Technology Planning Project of Guangdong Province of China under Grant 2017B010124003, Grant 2017B090909001, and Grant 2017B090909004. This article was recommended by Associate Editor J. Henkel. *(Corresponding author: Xiaoming Xiong.)*

X. Zheng, X. Hu, Y. Zhang, and X. Xiong are with the School of Automation, Guangdong University of Technology, Guangzhou 510006, China (e-mail: xinzheng9209@gmail.com; xianghong_hu@163.com; yz@gdut.edu.cn; xmxiong@gdut.edu.cn).

C. Xu is with the State Key Laboratory of Analog and Mixed-Signal VLSI, University of Macau, Macau, China (e-mail: chongyaox@gmail.com).

Digital Object Identifier 10.1109/TCAD.2019.2939330

On the other hand, hardware encryption has strong anti-attack capability but relatively higher cost [15]. To achieve a tradeoff between cost and performance [16], different co-design schemes and systems have been proposed [17]–[19], such as Fan–Vercauteren homomorphic encryption scheme [20], design space exploration of a hardware–software co-design [21], etc.

In the area of SW/HW co-design of crypto algorithms, Hafsa *et al.* [22] proposed an AES-ECC hybrid cryptosystem using a co-design approach which AES runs on NIOS II soft-core and ECC's scalar multiplication is implemented as a hardware accelerator. The implementation runs at a frequency of 157.63 MHz and consumes the power of 166.67 mW. It shows an interesting tradeoff between speed and area occupation. Sharif *et al.* [23] presented a hardware/software co-design of RSA cryptosystem that improves performance, while retaining flexibility. In this method, the performance versus flexibility tradeoff is investigated, and the result at 100 MHz shows a speedup of up to 57 times of co-design implementation versus the purely software implementation of RSA on the same platform. Xiao *et al.* [24] proposed an adaptive crypto system based on accelerators (ACSAs) with software and hardware co-design, which takes full advantages of both accelerators and CPUs for security HTTP accesses in big data. The user could adjust the tradeoff between CPU occupation and encryption performance through MM strategy, to free CPUs according to the working requirements. Basiri and Shukla [25] proposed an efficient hardware–software co-design for AES encryptor and RS-BCH concatenated encoder, where the latency and hardware cost lie in between the fully hardware and software-based designs. Feten *et al.* [26] offered a new hardware/software design flow using high-level synthesis and demonstrated the efficacy of this approach on the practical design example of AES-128. The gain reaches 11.76% for encryption operation and 12.4% for decryption operation compared to the pure software implementation. Dixit *et al.* [27] adopted the software and hardware co-design method to implement the AES cryptography algorithm on a soft core processor using Microblaze. This article was based on the study of the effectiveness of hardware–software co-design performance parameters.

In these proposed solutions, there is less details of the software and hardware co-design partitioning method given and the authors did not provide adequate structural analysis of the given algorithm modules [17]. Therefore, it is important to further study the efficient software and hardware co-design methods in SM2/3/4 cryptography.

In this article, we make the following contributions.

1) A new hybrid encryption/decryption and signature/verification flow using SM2/3/4 is proposed, in which the encrypted transmission and identity authentication are guaranteed.
2) Through the preliminary analysis of the pure software method, and considering the application scenario, hardware consumption, resource utilization, and the running time of each module, we analyze the reasons why each operation is implemented by software or hardware. And an optimal partitioning scheme is proposed.
3) The implementation of encryption/decryption and signature/verification is realized by this SW/HW co-design scheme. And the design of anti-simple power analysis (SPA) attacks is also added to the implementation of SM2 module.
4) Based on the SH/HW co-design solution for typical algorithm flow (SHT) and further analysis of the crypto algorithms, a parallel implementation of the SH/HW co-design solution of hybrid improvement algorithm flow (SHHI) is proposed to simultaneously process the encryption/decryption and signature/verification. This parallelization further improves the performance of the solution.
5) This design is taped-out on a silicon chip with SMIC 110-nm process. The experiment indicates that the proposed design achieves an effective tradeoff in terms of performance, power consumption, and area.

The rest of this article is organized as follows. Section II describes the proposed system design. Section III presents the solution of SW/HW co-design and parallel implementation. Section IV gives the experimental results and analysis. Section V concludes this article.

II. Proposed System Design

The hybrid cryptography flow using SM2/SM3/SM4 is given in Fig. 1. The encryption/decryption and authentication functions are realized in the flow above. The goal of this article is to efficiently implement such a hybrid cryptosystem via software and hardware co-design and fabricate a working chip.

A. Hybrid Cryptosystem

In this hybrid cryptosystem, SM4 is adopted to encrypt the plaintext of large data [6], and the key is encrypted by SM2. Its block length and cipher key length are both 128 bits. The structure of decryption is the same as the encryption, and the round keys of decryption are in the reverse order of encryption. The encryption algorithm first iterates the round function F for 32 times, and then applies the reverse transformation R in the end. Suppose its input plaintext is $(X_0, X_1, X_2, X_3) \in (Z_2^{32})^4$. The corresponding output ciphertext is $(Y_0, Y_1, Y_2, Y_3) \in (Z_2^{32})^4$, and the round keys are $rk_i \in Z_2^{32}$, $i = 0, 1, \ldots, 31$. Then, the process of the encryption algorithm is as follows.

Firstly, the 32-round iterative operation is executed

$$X_{i+4} = F(X_i, X_{i+1}, X_{i+2}, X_{i+3}, rk_i), i = 0, 1, \ldots, 31. \quad (1)$$

Secondly, the reverse transformation operation is carried out

$$\begin{aligned}(Y_0, Y_1, Y_2, Y_3) &= R(X_{32}, X_{33}, X_{34}, X_{35})\\ &= (X_{35}, X_{34}, X_{33}, X_{32}). \quad (2)\end{aligned}$$

In order to obtain the digital signature by a signer and verify the validity of the signature by a verifier, SM2 is used [5]. Sender A owns the identifier ID_A with the length of $ENTL_A$. And the key pair of A includes a private key d_A and a public key $P_A = [d_A]G = (x_A, y_A)$, a base point $G = (x_G, y_G)$ is on the elliptic curve with prime order. Before signature and verification processes, the message should be compressed using SM3. Thereinto, a and b are the parameters of the elliptic curve equation. Both signer and verifier need to obtain $Z_A = \mathrm{SM3}(\mathrm{ENTL}_A||\mathrm{ID}_A||a||b||x_G||y_G||x_A||y_A)$

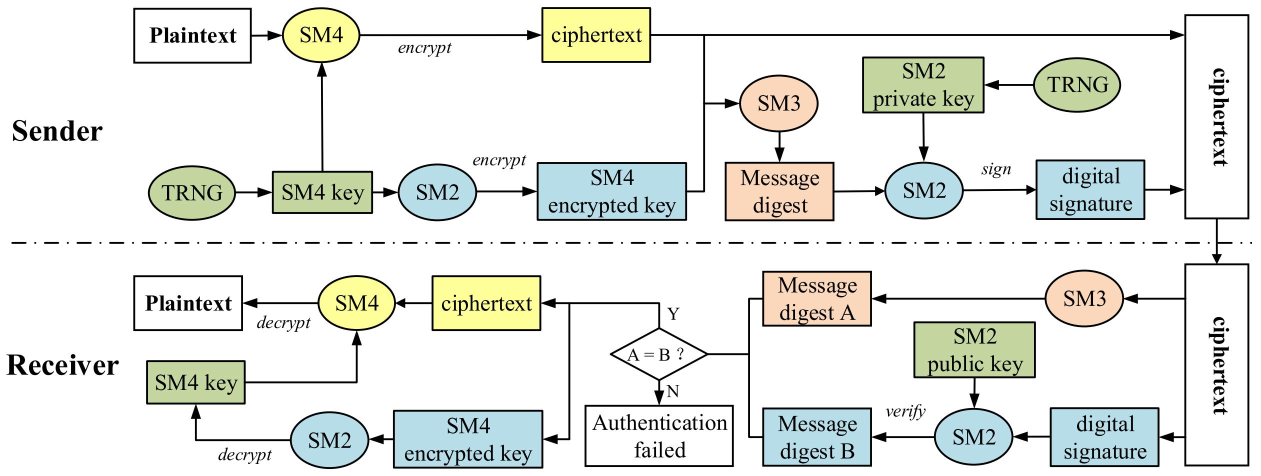


Fig. 1. System function block diagram.

by calculating the hash value of Sender A. If Z_A is not the hash value corresponding to Sender A, the verification cannot be passed through. Moreover, the private key of SM2 and the SM4-key are generated by the true random number generator (TRNG) [28].

B. Software-Only Analysis and SW/HW Partition

The premise of software and hardware co-design is to implement the module which occupies more resources in hardware and the others in software. Additionally, operations which are used repeatedly in an algorithm should be implemented in hardware and high-level control structures in software. In order to achieve the optimal partition of software and hardware, pure software simulation is needed. Based on the Miracle library [29] which provides the underlying operations in many cryptographic algorithms, the SM2/3/4 algorithm can be constructed. Since the requirement of the supporting project, the CPU is fixed to CK802 of C-Sky (a 32-bit embedded low-power CPU). In order to ensure the reproducibility, the pure software scheme is not only implemented on CK802 but also on a more standard 32-bit embedded microcontroller ARM Cortex-A9. The experimental results of both implementations are presented. For the CK802, which has an operating frequency of 36 MHz, C/C++ is used for implementation on C-Sky provided platforms. For the ARM Cortex-A9 with operating frequency of 666 MHz, the pure software design is implemented on the Xilinx Zynq-7000 SoC platform. And the running time of each module is measured for 1000 times, then the average results are shown in Tables II–IV.

According to Table II, the point multiplication (PM) accounts for the largest running time of SM2 digital signature. Thus, PM is the performance bottleneck. Because PM will repeatedly call the modular multiplication (MM), the basic arithmetic unit modules, including point operations and modular operations can be implemented in hardware. In Table III, the longest occupation time of SM3 is the expanding and compressing part, which exceeds 92% of the entire hash time. Therefore, the input data can be first grouped and filled by software. Then, the hardware module is called to expand, iterate, and compress, and finally, a hash value is generated [30]. Similarly, as shown in Table IV, key expansion, encryption, and decryption occupy the highest proportion in SM4, so these parts can be implemented by hardware. There is a strong

TABLE II
RUNNING TIME OCCUPATION RATIO OF SM2 DIGITAL SIGNATURE

Function module		Run time(ms)		Occupation(%)	
		C-Sky	ARM	C-Sky	ARM
Preprocess	preprocess 1	0.48	0.08	0.03	0.04
	key-pair generation	314.68	52.45	24.56	25.04
Signature	preprocess 2	0.16	0.03	0.01	0.01
	$(x_1, y_1) = [k]G$	331.44	51.24	25.87	24.46
	$r = (e + x_1) mod n$	5.43	0.84	0.42	0.39
	$s = ((1 + d_A)^{-1} * (k - r * d_A)) mod n$	21.44	3.57	1.68	1.70
Verification	preprocess 2	0.16	0.03	0.01	0.01
	$t = (r' + s') mod n$	4.12	0.69	0.32	0.33
	$(x_1', y_1') = [s']G + [t]P_A$	598.95	99.82	46.74	47.65
	$R = (e' + x_1') mod n$	4.45	0.74	0.34	0.35

TABLE III
RUNNING TIME COMPARISON OF EACH MODULE OF SM3

Operations	Run time(us)		Occupation(%)	
	C-Sky	ARM	C-Sky	ARM
Grouping and Padding	15.23	2.20	6.77	7.04
Expanding	79.76	14.86	35.48	47.58
Compressing	129.86	14.17	57.75	45.38

TABLE IV
RUNNING TIME COMPARISON OF EACH MODULE OF SM4

Operations	Run time(us)		Occupation(%)	
	C-Sky	ARM	C-Sky	ARM
Message Grouping	3.26	0.12	0.21	0.21
Key Expansion	267.71	10.28	16.96	18.38
Encryption	652.59	22.74	41.34	40.66
Decryption	654.89	22.79	41.48	40.75

connection between the identified performance bottlenecks in Tables II–IV and the target software platform.

By modeling and analyzing the execution time of the algorithms and the number of times the module is called in the process of program execution, the preliminary partitioning results are obtained. The results are further adjusted by considering the application scenario of low-power smart devices, the hardware overhead, resource utilization, and performance requirements.

For the SM2 module, the implementation of elliptic curve algorithm needs to complete three layers of operations. The first layer is the modular operation on the prime field, including modular addition (MA), modular subtraction (MS), MM, and modular inverse (MI). The second layer is the point operation on the elliptic curve, including the point addition (PA), multiple point (MP) and PM, in which PM is the core operation. The third layer is to complete the process of the specific cryptographic algorithm which can completely realize the related applications. We found out that PM operations not only occupy more than half of the execution time but also require continual calling of the MM. If the PM is only deployed in software and MM in hardware, the hardware and software will generate a large number of intermediate variables and data interaction. What is more, the interaction with CPU will occupy a lot of running time, thus affect the performance. All the modular operations constitute the basic unit of the point operations, and the point operations constitute the arithmetic unit of the algorithm. Since the algorithm flow requires high flexibility and modifiability, we implement the basic arithmetic unit modules, including point operations and modular operations in hardware and the signature/verification algorithm flow in software to achieve system performance balance.

For SM3, the measurement function requires multiple state machines and is complex to implement since the size of the input data is variable. The grouping and padding are not repeatedly used in the encryption process and the hardware implementation consumes a lot of resources. We decide to implement these operations in software to save hardware resources. For expansion and compression, multiple iterations are required. Thus, these repeatedly used modules are more suitable for hardware implementation. SM4 is similar to SM3 in partitioning mechanism. Data grouping is implemented by software. Round-key generation and encryption/decryption with multiple iterations are implemented in hardware.

Overall, the most computationally intensive parts of the system are implemented by hardware. The application layer and each module's configuration, control, data reading, etc., are realized by software. The partitioning based on the Miracle library is used the same instruction set. This solution saves the logic resources, enhances the efficiency without affecting system performance, and retains the portability compared with the solution implemented by pure hardware. The additional overhead mainly comes from processor instruction calls and data transfers, but these effects are very limited. Table V is the specific functional division of hardware and software modules.

TABLE V
FUNCTIONAL DIVISION OF HARDWARE AND SOFTWARE MODULES

Module	Hardware	Software
SM2	Modular Operations, Point Operations	Signature Generation, Signature Verification, Mode Control, Data Processing
SM3	Expansion, Compression	Grouping and Padding, Mode Control, Data Processing
SM4	Round Key Generation, Encryption, Decryption	Grouping, Mode Control, Data Processing

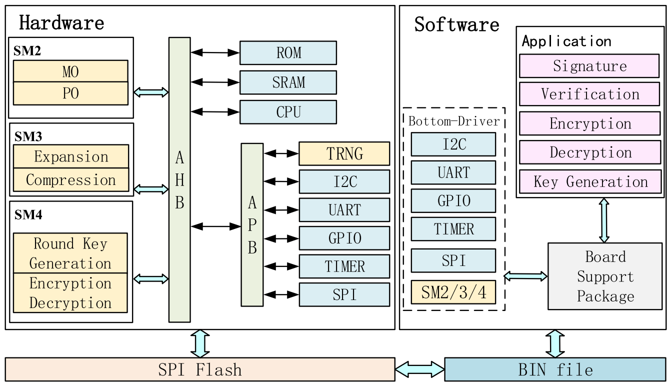


Fig. 2. System structure diagram.

C. Co-Design Architecture of the SoC System

The SoC system as shown in Fig. 2 consists of a 32-bit embedded processor CK802 with CPU frequency of 36 MHz, AMBA high-performance bus (AHB) and AMBA peripheral bus (APB) interface, the hardware modules of SM2/SM3/SM4/TRNG, 8 kB read-only memory, and 128 kB static random access memory which is used to store temporary data in the SoC system. The AHB bus interface control logic units and four kinds of registers [control registers (CRs)/status registers (SRs)/input registers (IRs)/output registers (ORs)] make up the AHB bus interface IP. This AHB IP is used in SM2/3/4 and other modules to configure the operating mode or record the operating state. The proposed system also includes the general interface modules, such as SPI, IIC, GPIO, UART, and timer module. The software part mainly controls the entire system by calling hardware resources and realizes the software/hardware scheduling to complete the signature/verification and encryption/decryption. The SM2/3/4 hardware modules are high-performance modules which perform in realtime and they are directly mounted on the AHB bus [31]. The low-speed modules and interfaces, such as TRNG, SPI, IIC, GPIO, and UART, are connected through APB bridge of AHB to communicate with the CPU.

1) Hardware Module Implementation of SM2/3: The SM2 hardware module implementation is based on the elliptic curve over $GF(p)$. The SM2 module consists of the blocks as follows: MA, MS, MM, MI, and PA, MP, PM. In order to reduce

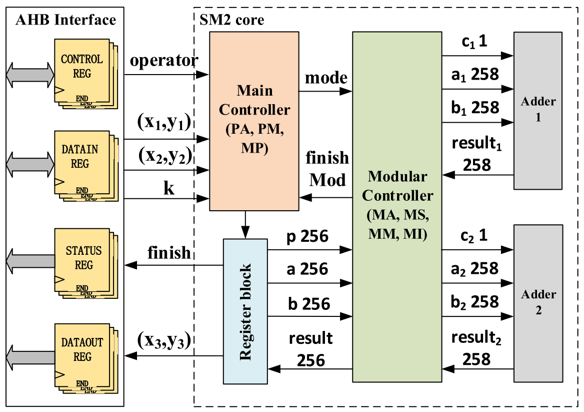


Fig. 3. Hardware architecture of SM2.

Algorithm 1 Interleaved MM Algorithm

```
Input: $X, Y, M$ (n bit), $X < M, Y < M, M \approx 2^n$
Output: $P = X * Y mod\ M$
1: $P = 0$
2: for $i = n - 1$ downto 0 do
3:     $P = 2P + X_i * Y$
4:     $P = P - (P_{n+1}, P_n) * M$
5: if $P > M$ then
6:     $P = P - M$
7: return $P$
```

the chip area and power consumption, the main hardware overhead of SM2 is two 256-bit adders, four 256-bit registers, and several 256-bit multiplexers. The hardware architecture of SM2 is illustrated in Fig. 3. The main controller is a state machine which controls PA, MP, and PM and data transforming between the register block and the modular controller. The modular controller performs one of MA/MS/MM/MI at a time.

MA/MS is implemented by two steps of addition and subtraction operations. MA/MS can be done in only one cycle, so we adopt two full-word adders. For example, a, b, and p are inputs. $0 \leq a, b < p$. p is a prime. For MA, if $a + b \geq p$, the result is $a+b-p$. Otherwise, the result is $a+b$. An improved interleaved MM algorithm is used in MM as described in Algorithm 1. Comparing with the traditional Montgomery or specific prime field MM [32], this MM algorithm uses only two adders and one full-word register in one iteration and avoids the timing latency of comparison.

In the work of [33], both MI and MM operations can be achieved in the same run time. In order to make sure each step can be completed in one cycle, three adders and a comparator are needed. In our proposed design, the MI adopts the binary Euler inversion algorithm that consumes minimum hardware resource. Algorithm 2 presents the details. The r and s are both in the range of $[0, p)$. The range of $r-s$ is $(-p, p)$. In order to ensure that r, s still within the range after modulo operation, there are three different situations: 1) if the difference of r and s is odd, $r = r/2 \bmod p$ can be achieved by adding p and shifting one bit; 2) if the difference is negative even, adding $2P$ and shifting one bit; and 3) for all other cases, just shifting

Algorithm 2 Binary Euler Inversion Algorithm

```
Input: $a, b, p, 0 < a < p, 0 < b < p, p$ is a prime
Output: $b * a^{-1} mod\ p$
1: $u = p,\ v = a,\ r = 0,\ s = b$
2: while ($u \neq 1$ and ($v \neq 1$) do
3:     if $u[0] == 0$ then
4:         $r = r/2\ mod\ p,\ u = u/2$
5:     else if $v[0] = 0$ then
6:         $s = s/2\ mod\ p,\ v = v/2$
7:     else if ($u[0] == 1$) and ($v[0] == 1$) then
8:         if $u > v$ then
9:             $r = r - s,\ u = u - v$
10:        else
11:            $s = s - r,\ v = v - u$
12: if $u == 1$ then
13:     return $r\ mod\ p$
14: else
15:     return $s\ mod\ p$
```

one bit. In this way, the loop process of step 2 is guaranteed to converge and r, s does not overflow.

Overall, by sharing the adder and multiplexing register resources, MI requires only two adders and four registers of hardware overhead. MA/MS can be completed in one clock cycle. The MM requires n cycles. And the MI needs slightly more than $2n$ cycles.

Since the operations of MA/MS/MM/MI are using the same adders, these operations must be performed step by step. The overhead of MI is about twice the cost of MM. This design directly uses the affine coordinates instead of introducing projection coordinates. Suppose p is a prime greater than 3 and the elliptic curve equation over finite-field F_p in the affine coordinate system has the simplified form as $y^2 = x^3+ax+b$, where $a, b \in F_p$, satisfying $(4b^3+27b^2) \bmod p \neq 0$. The set of points on the elliptic curve is denoted by $E(F_p) = \{(x, y)|x, y \in F_p, y^2 = x^3+ax+b\} \cup \{O\}$, where O is the point at infinity. PA and MP are consistent for the expression of the given slope of a line

$$\begin{cases} x_3 = k^2 - x_1 - x_2 \\ y_3 = k(x_1 - x_3) - y_1 \end{cases} \tag{3}$$

where

$$k = \begin{cases} \frac{y_2-y_1}{x_2-x_1}, & \text{if } x_1 \neq x_2 \\ O, & \text{if } x_1 = x_2, y_1 \neq y_2 \\ \frac{3x_1^2+a}{2y_1}, & \text{if } x_1 = x_2, y_1 = y_2. \end{cases} \tag{4}$$

Therefore, when scheduling, the k can be calculated. The same scheduling can be shared when the x_3 and y_3 are calculated. In this article, PA requires two MM and one MI operations and the MP requires three MM and one MI operations. The number of state machines can be controlled under 16 or less.

Because PM which is performed by the binary expansion algorithm is easily cracked by the SPA. The PM needs to be able to resist SPA attacks. Therefore, the SPA resistant PM algorithm is used as described in Algorithm 3, where

Algorithm 3 SPA Resistant PM Algorithm

```
Input: elliptic curve point $P$, scalar $k$
Output: elliptic curve point $Q[0]$ : $Q[0] = kP$
1: $Q[0] = P$
2: for $i = n - 1$ downto 0 do
3:     $Q[0] = 2Q[0]$
4:     $Q[1] = Q[0] + P$
5:     $Q[0] = Q[k_n]$
6: return $Q[0]$
```

$k = (k_{n-1}, \ldots, k_0)$ and n is the binary length of k. This algorithm performs PA and MP regardless of whether the private key bit is 0 or 1.

The SM3 hardware module, given in Fig. 4, is implemented in two parts: 1) expansion and 2) compression [30]. Through the analysis of SM3 in [6], the message is expanded into 132 32-bit words, which requiring the first 68-round of iteration $w_0, w_1, \ldots, w_{67}$ and the second 64-round of iteration $w'_0, w'_1, \ldots, w'_{63}$. However, the data required for the second iteration does not need to wait until the first iteration are completed. Thus, the second iteration can be nested into the first iteration and begin with its fifth round. In this way, only 68 rounds of iterations, or 68 cycles, are required. Compression requires a 64-round loop iteration. Similarly, compression can be nested with the expansion process, starting with its sixth round. In the end, it takes only 69 cycles to complete an expansion and compression by using this scheduling. Obviously, the theoretical performance will reach an increase of nearly 2.84 times if not considering the overhead of other circuits.

We also adopt the register multiplexing strategy based on this scheduling. The expansion module needs 18 32-bit registers (16 for storing intermediate data, and 2 for saving current w_i, w_j for compression). The compression module requires 16 32-bit registers (8 for storing the initial value of 256-bit or the result of the previous 512-bit data block $B^{(i)}$ and 8 for storing the currently newly acquired 256-bit data). Thus, a total of 34 32-bit registers are required. Finally, only 34 32-bit registers are needed to complete an expansion and compression. If there is no scheduling optimization, $132 + 16 = 148$ 32-bit registers are required. Obviously, the number of registers has dropped by nearly 4.35 times.

2) Hardware Module Implementation of SM4: The SM4 hardware modules are composed of two parts [6]. The first part is the round-key generation and second part is encryption/decryption. In Fig. 5, the architecture of SM4 is depicted. The block length and cipher key length are both 128 bits. SM4 adopts an unbalanced Feistel structure and iterates its round functions for 32 times in both encryption and key expansion algorithms. The round-key generation module is actually the logic implementation of the secret key expansion algorithm. The structure of decryption is the same as the encryption. But the decryption round keys are in the reverse order of the encryption round keys.

To store the large number of constants in the SM4 algorithm, SBoxes of $16 * 16 * 8$ bits, an FK of $4 * 32$ bits, and a CK of $32 * 32$ bits are used in the implementation. SBoxes consume

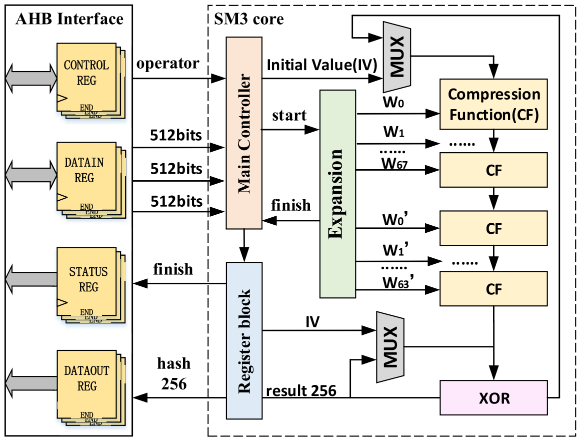


Fig. 4. Hardware architecture of SM3.

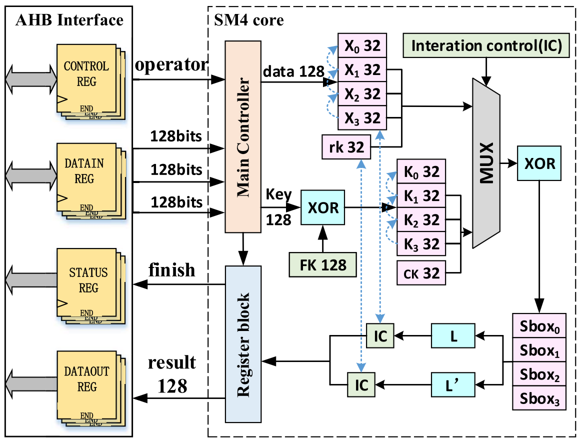


Fig. 5. Hardware architecture of SM4.

the most of the storage. There are four SBoxes in the nonlinear transformation τ, where $A = (a_0, a_1, a_2, a_3) \in (Z_2^8)^4$ is the input to τ and $B \in (Z_2^8)^4$ is the corresponding output

$$\begin{aligned} B &= (b_0, b_1, b_2, b_3) = \tau(A) \\ &= (\text{Sbox}(a_0), \text{Sbox}(a_1), \text{Sbox}(a_2), \text{Sbox}(a_3)). \end{aligned} \tag{5}$$

The generation of one round key and encryption can be completed within one clock. Although the four SBoxs take up certain resources, the control logic is simple. The linear transformations L and L' are composed of cyclic shift and XOR

$$\begin{aligned} L(B) &= B \oplus (B <<< 2) \oplus (B <<< 10) \\ &\quad \oplus (B <<< 18) \oplus (B <<< 24) \\ L'(B) &= B \oplus (B <<< 13) \oplus (B <<< 23). \end{aligned} \tag{6}$$

Because the number of cyclic shifts is fixed, the combinatory logic can be adopted to build a dedicated shift circuit which is simple and fast. The value of the parameter CK is fixed, so it can be stored directly. Overall, the data encryption/decryption process takes 35 clock cycles.

3) TRNG Module: In order to generate high-speed true random numbers or pseudorandom numbers [34], TRNG is used. It consists of a high entropy true random source, a post

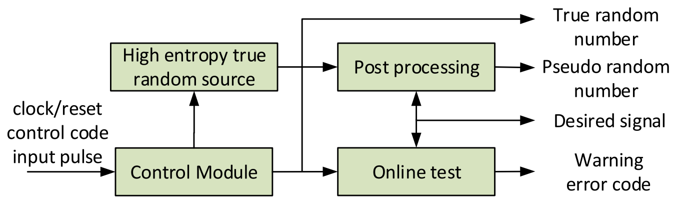


Fig. 6. TRNG module diagram.

processing, and an online test module. Depending on the configuration, this ring oscillator (RO)-based TRNG can generate completely nondeterministic random numbers for high security applications. The TRNG module also has an interface module connected to the APB bridge. It is not necessary to input an excitation signal, directly configure the CR to select the corresponding mode to start, and obtain a random number from the OR. The structure of the TRNG module is illustrated in Fig. 6.

The high entropy true random source consists of 32 ROs. Sample clock comes from control module, which is generated by dividing system clock. In this design, ROs are sampled with the frequency of 1/4 system clock frequency.

III. Software/Hardware Co-Design

In our design, all the modular operations and point operations are implemented in hardware and scheduled by software. The hardware parts do not implement the functions of signature/verification, but they are scheduled and executed by software. By doing so, hardware saving and resource sharing are both achieved.

A. Improved Implementation of SW/HW Co-Design

Since the PM operation of SM2 hardware module consumes the most resources and time, it is possible to execute CPU and other parallel hardware operations while performing PM. And the PM operation is decomposed into startup module, execution module, and output module. The optimization points of the design algorithm are as follows.

1) During the SM2 public key generation($P_A = [d_A]G = (x_A, y_A)$) process, first, the startup module of PM is executed, and the TRNG module is simultaneously called to perform the key generation of SM4 and the random number generation, and then read the result from the PM output module.
2) After initiating the startup module of PM to calculate the point on the elliptic curve($(x_1, y_1) = [k]G$), perform both SM4 encryption and SM2 preprocessing which includes calculating the hash value twice of the message via SM3, and then read the PM result.
3) The verification process is optimized in the same way as the signature process, and the SM2 preprocessing or SM4 decryption is executed while performing the PM operation. The explicit algorithm optimization steps are detailed in Algorithms 4 and 5.

The improved algorithm combines SM4 encryption/decryption and SM2 signature algorithm. Based on the previous software/hardware partitioning and analysis of

Algorithm 4 Hybrid Encryption and Signature Algorithm

```
1: Generate SM2 private key $d_A = TRNG()$;
2: Initiate the startup module of PM to compute $P_A = [d_A]G$;
3: Generate a random number $k \in [1, n-1]$ and the key of
   SM4 $sm4\_key$ using TRNG;
4: Read the $P_A$ from the PM output module;
5: Initiate the startup module of PM to compute $(x_1, y_1) =$
   $[k]G$;
6: Execute SM4 encryption;
7: Set $\overline{M_0} = Z_A \parallel M_0$ and compute $e = SM3(\overline{M_0})$;
8: Read the $(x_1, y_1)$ from the PM output module;
9: Compute $r = (e + x_1) mod n$;
10: **if** $r = 0$ or $r + k = n$ **then**;
11:     return step5;
12: Compute $s = ((1 + d_A)^{-1} * (k - r * d_A)) mod n$;
13: **if** s=0 **then**
14:     return step5;
15: The signature of message $M_0$ is (r, s).
```

Algorithm 5 Hybrid Decryption and Verification Algorithm

```
1: Initiate the startup module of PM to compute $(\overline{x_1'}, \overline{y_1'}) =$
   $[s']G$;
2: Execute SM4 decryption;
3: Verify whether $r' \in [1, n-1]$ holds;
4: **if** $r' \notin [1, n-1]$ **then**
5:     verifier output reject;
6: Verify whether $s'$ in $[1, n-1]$;
7: **if** $s' \notin [1, n-1]$ **then**
8:     verifier output reject;
9: Set $\overline{M_0'} = Z_A \parallel M_0'$ and compute $e' = SM3(\overline{M_0'})$;
10: Read the $(\overline{x_1'}, \overline{y_1'})$ from the PM output module;
11: Compute $t = (r' + s') mod n$;
12: **if** $t = 0$ **then**
13:     verifier output reject;
14: Compute the point $(x_1', y_1') = (\overline{x_1'}, \overline{y_1'}) + [t]P_A$;
15: Compute $R = (e' + x_1') mod n$;
16: **if** $R = r'$ **then**
17:     verifier output accept;
18: **else**
19:     verifier output reject;
```

the performance of each module and taking full advantage of parallel execution of CPU and hardware, the proposed improved algorithm reduces the running time significantly. The process of random number generation and key generation are also included in the above improved algorithms to further improve the operation efficiency. The input message M_0, which is to be signed, consists of ciphertext and SM4 encrypted key. SM4 encryption and SM2 signature are performed simultaneously. SM4 decryption and SM2 verification are also performed simultaneously. The analysis of SM2 encryption/decryption algorithm is not included here, but it is used to encrypt/decrypt the key of SM4 in order to guarantee the security of SM4 key.

B. Scheduling

Fig. 7 depicts the scheduling process of encryption and signature. The red boxes represent the software portions that schedules the corresponding hardware operations which are represented by the green boxes. The inputs and outputs are represented by the yellow boxes.

From the upper left of the activity diagram, the TRNG module can be called by software to generate random numbers which are used as the private key of SM2, the random numbers in the signature process, and the key of SM4. For the right side of the activity diagram starting from the input message, the TRNG module is first called to generate the random numbers as the private key of SM2. Hardware modules communicate with the software through the AHB interface and the software reads the current statuses and controls the start and stop of the hardware modules. When software issues the command, the corresponding hardware performs the PM operation and generates the public key. Because the TRNG module and the PM module are independent of each other, the random number k and the SM4 key can be generated simultaneously. This is the first parallel calculation. After the PM calculation is completed, the software reads the result of the PM as the public key and performs the next calculation. The second dotted line box in the diagram represents the second parallel calculation. As shown in Fig. 7, the second PM operation, the SM4 encryption operation, and the two preprocessing operations in the signature process are performed simultaneously. After the second PM operation is completed, the output of the PM is read by software and then the other operations in Algorithm 4 are performed until the signature is generated.

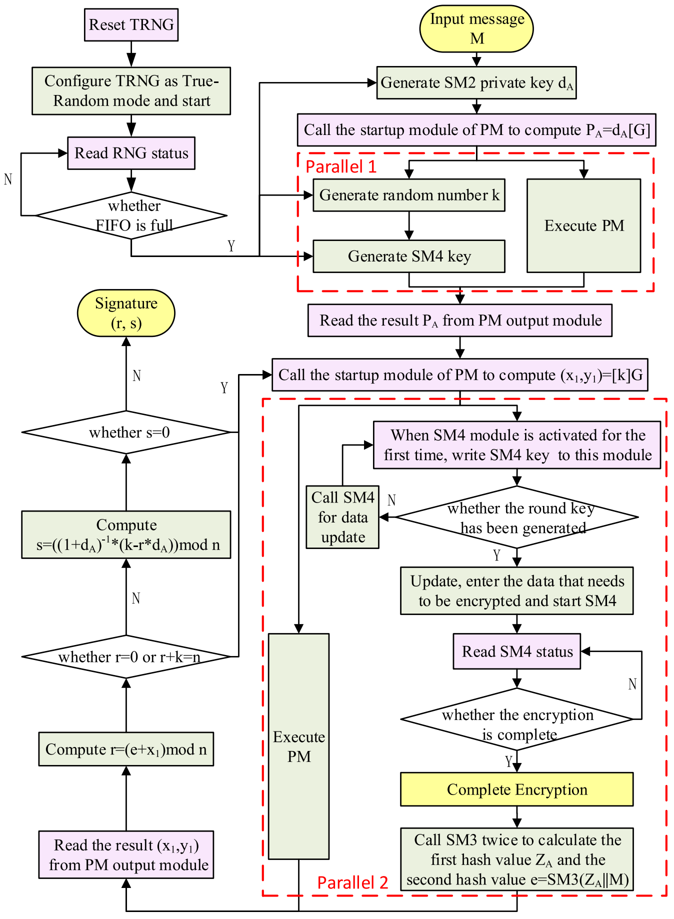
After scheduling steps, the ciphertext and the digital signature can be received. The scheduling of verification and decryption is similar to signature and encryption. The receiver first computes the digest of ciphertext through the SM3 algorithm and compares it with the digest which is verified by SM2. If these two digests are the same, the receiver then can decrypt the ciphertext by SM4 to get the plaintext.

IV. Experiments

In this section, we will investigate the performance of our approach and implementation. For comparison, another two SM2/3/4 encryption/decryption and signature/verification methods are used as baselines. The second baseline is the pure software solution, which is a typical software cryptosystem with low security and efficiency. The third one is typical software/hardware co-design system, which has no improvement of the algorithm. Actually, we also compare the performance of our single module with others [22], [35]–[37]. The performance of our design is generally evaluated based on run time and power consumption.

The system clock frequency of $f = 36$ MHz is used in the experiments. If the length of input message M is len and the clock period for one round of encryption or decryption is T, the throughput of SM4 encryption and decryption is as follows:

$$V_{sm4} = \frac{f \cdot \mathrm{len}}{T \cdot 1024 \cdot 1024}. \tag{7}$$



Reset TRNG
Configure TRNG as True-Random mode and start
Read RNG status
N
whether FIFO is full
Y
Input message M
Generate SM2 private key d_A
Call the startup module of PM to compute $P_A=d_A[G]$
Parallel 1
Generate random number k
Execute PM
Generate SM4 key
Read the result P_A from PM output module
Signature (r, s)
N
Y
whether s=0
Call the startup module of PM to compute $(x_1,y_1)=[k]G$
Compute $s=((1+d_A)^{-1}*(k-r*d_A))$mod n
N
whether r=0 or r+k=n
Compute $r=(e+x_1)$mod n
Read the result (x_1,y_1) from PM output module
When SM4 module is activated for the first time, write SM4 key to this module
Call SM4 for data update
N
whether the round key has been generated
Y
Update, enter the data that needs to be encrypted and start SM4
Read SM4 status
N
whether the encryption is complete
Y
Complete Encryption
Execute PM
Call SM3 twice to calculate the first hash value Z_A and the second hash value e=SM3(Z_A‖M)
Parallel 2

Fig. 7. Process of scheduling to realize encryption and signature.

According to the standard SM2 digital signature algorithm flow [5], the generation of public key requires one PM and one SM3 operation is need for preprocessing. The signature process requires one PM, one SM3 operation, and multiple modular operations. Meanwhile, the verification needs two PM, one PA, one SM3 operation, and two modular operations. Suppose the length of input M_0 is len_0, and $Z_A||M_0 = 256 + len_0$. λ_1 and λ_2 are, respectively, the numbers of clock cycles required by the CPU for XOR operation, data splicing, data transfer, and scheduling control in the process of signature and verification. T_{pm} and T_{sm3} are the numbers of clock cycles needed for PM and SM3, respectively. Hence, the theoretical signature and verification rate can be calculated by (8) and (9), respectively.

$$V_{\mathrm{sign}} = \frac{f}{T_{pm} + \frac{Z_A||M_0}{512} \cdot T_{sm3} + T_{mo} + \lambda_1} \tag{8}$$

where T_{mo} is the number of clock cycles required for the MO, and an MO includes three MA, two MM, one MS, and one MI, that is, $T_{mo} = 3 \cdot T_{ma} + T_{ms} + T_{mi} + 2 \cdot T_{mm}$.

$$V_{\mathrm{ver}} = \frac{f}{2 \cdot T_{pm} + \frac{Z_A||M_0'}{512} \cdot T_{sm3} + T_{pa} + 2 \cdot T_{ma} + \lambda_2} \tag{9}$$

The ASIC designs of SM2, SM3, and SM4 are done with SMIC 110-nm process with a system clock of 36 MHz. A

TABLE VI
PERFORMANCE COMPARISON OF EACH MODULE

Operations	**Software-only**		**SHT**		**SHHI**	
	C-Sky	**ARM**	**C-Sky**	**ARM**	**C-Sky**	**ARM**
MA/MS	4450 us	694 us	22 us	11 us	22 us	11 us
MM	5224 us	753 us	30 us	17 us	30 us	17 us
MI	6725 us	813 us	41 us	22 us	41 us	22 us
PA	8285 us	901 us	75 us	49 us	75 us	49 us
PM	331440 us	51243 us	10964 us	9255 us	10964 us	9255 us
SM3	225 us	32.23 us	133 us	32 us	133 us	32 us
TRNG	-	-	69 us	59 us	69 us	59 us
SM4	0.35 Mbps	2.18 Mbps	12.36 Mbps	13.40 Mbps	12.36 Mbps	13.40 Mbps
Public key generation	3.18 times/sec	18.37 times/sec	77.82 times/sec	102.51 times/sec	77.83 times/sec	102.53 times/sec
Signature Generation	2.79 times/sec	16.75 times/sec	69.59 times/sec	100.83 times/sec	83.01 times/sec	107.42 times/sec
Verification	1.66 times/sec	9.12 times/sec	36.76 times/sec	50.14 times/sec	42.99 times/sec	52.61 times/sec
Overall Program	1280146 us	209601 us	55633 us	38461 us	48765 us	37355 us

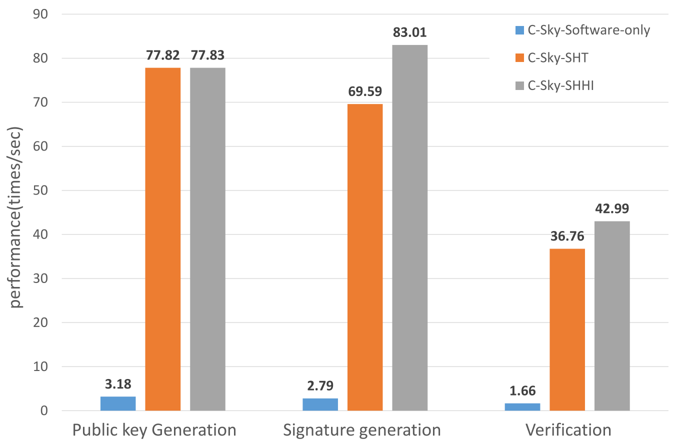


Fig. 8. Performance comparison of using C-Sky.

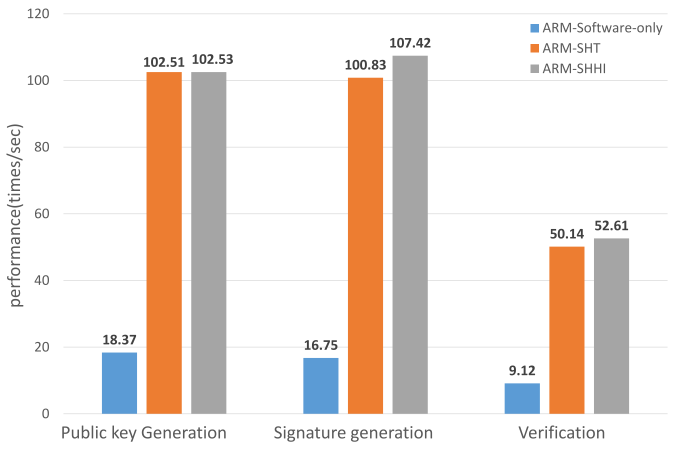


Fig. 9. Performance comparison of using ARM.

timer module mounted at APB bus contains four independently programmable 32-bit counters which is implemented to test the performance of individual modules. The length of input message used here is 1024 bits. For the ASIC with C-Sky, the performance comparison of three kinds of solutions, software-only, SHT, and SHHI, are shown in Table VI and Fig. 8. The hardware interface clock frequency of crypto module is 50 MHz on field-programmable gate array (FPGA) and the corresponding performance results are shown in the columns of ARM in Table VI and Fig. 9.

As shown above, after SW/HW division and co-design, for the scheme of C-sky, SHT significantly outperforms software-only method and achieves roughly 35 times speed-up in terms of SM4 throughput and the public key generation frequency also increases 75 times/s. In terms of signature/verification, the number of signature generation has increased by about 67 times/s and the number of verification has increased by 35 times/s. For the ARM implementations, SHT outperforms software-only solution. It achieves roughly 6 speed-up of SM4 throughput and generates public keys 84 more times/s. The number of signature generation has increased by about 84 times/s and the number of verification has increased by 41 times/s. Via the algorithm improvement and performance optimization after software and hardware co-design, SHHI has further improved the performance compared with SHT as shown in Table VI. Finally, the experiments using these three schemes are conducted to compare the complete encryption/decryption and signature/verification process of this system. The overall performance of SHHI is 12.35% and 2.9% higher than SHT, which is 96.19% and 82.18% higher than software-only method, respectively, in the scheme of C-Sky and ARM.

Table VII shows the SHT and SHHI scheme compared to four similar work. In order to achieve a more reasonable comparison result, when different clock frequencies are used in different schemes, we adopted approximate equivalent method to normalize the clock frequency to 36 MHz in Table VII. The item of SM4/AES represents the throughput of SM4 or AES encryption and decryption; the item of SIGN indicates that the implemental time of a signature can be generated when signing per message; and the item of VERIFY represents that the implemental time of once verification can be executed when verifying per message. The scheme 1 is the implementation results of the AES algorithm in the PSoC at 50 MHz [22]. Scheme 2 shows the AES algorithm which is implemented on Cyclone II FPGA-based around NIOS II processor [35]. Because the AES algorithm in schemes 1 and 2

TABLE VII
COMPARISON WITH PREVIOUS WORKS

Scheme	Platform	Operation	FPGA Resouce		Performance
			Slice LUTs	Slices	
[22]	Cyclone IV.E	AES	-	-	1.17Kbps
[35]	Cyclone II	AES	-	-	0.83Kbps
[36]	Virtex 5	SIGN	13664	3416	0.13s
		VERIFY	13664	3416	0.14s
[37]	MSP430	SIGN	-	-	0.30s
		VERIFY	-	-	0.44s
[38] (w=18)	Virtex 5	SIGN	29250	11040	1.02ms
		VERIFY	34177	12846	1.09ms
[38] (w=9)	Virtex 5	SIGN	23675	8773	1.71ms
		VERIFY	27791	9967	1.87ms
SHHI-CSky	SmartL	SIGN	16195	7473	12.04ms
		VERIFY	16195	7473	23.26ms
		SM4	12894	8023	12.36Mbps
SHHI-ARM	Zynq-7000	SIGN	13821	6297	12.84ms
		VERIFY	13821	6297	26.23ms
		SM4	10086	6014	9.71Mbps

TABLE VIII
COMPREHENSIVE PERFORMANCE INDICATORS OF EACH MODULE

Scheme	Module	Gates	Area(um^2)
SW/HW co-design	SM2	56128	285815
	SM3	18087	92100
	SM4	124860	635812
pure HW	SM3	19300	98281
	SM4	136810	696667

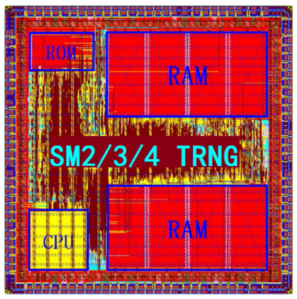


Fig. 10. ASIC layout.

are implemented by software, the throughput of our design is higher than both two schemes. Scheme 3 implements an efficient software/hardware cooperative SoC for SM2 digital signature with the Micro Blaze as master processor and a TTA-like ECC coprocessor at 75 MHz, but the generation of hash value consumes a lot of time [36]. Although the FPGA resource of scheme 3 is less than our work, the signature performance of our design is roughly 10.8 times better than that of scheme 3. Scheme 4 costs 1.35 s with an 8 MHz, 16-bit CPU for 160-bit ECC to generate a signature and 1.96 s to perform a signature verification [37]. The signature/verification performance of schemes 5 and 6 is better than that of SHHI-CSky and SHHI-ARM, but the resource utilization of Slice LUTs and Slices is much more than that of our design. Moreover, implementations in [38] are over $GF(2^{163})$ in which the complexity of PM is lower than $GF(p)$. For the signature/verification, the results show that the performance of SHHI outperforms schemes 3 and 4. And the resource utilization is less than schemes 5 and 6. For the encryption/decryption, the performance of SHHI outperforms schemes 1 and 2. Thus, our design has an optimal tradeoff between performance and area.

On the other hand, because PA and MP operations are performed for every bit of the scalar, side-channel attacks can be resisted by the modification we made in Algorithm 3. It behaves the same on the power waveform. The private key cannot be read directly from the power curve. Since each round of loops requires PA and MP calculations, both time overhead and resource consumption are increased. Except for SPA, other type of attacks, such as MOV, anomalous curve, Pohlig–Hellman, and Pollard attack are also considered. The elliptic curve we used, referring to [5], satisfies the condition for resisting these attacks. At the algorithm level, key randomization or insertion redundancy is also a better strategy to defend the side-channel attacks.

The power consumption of this designed chip is also tested and analyzed with the statistical analysis method to measure the static and dynamic power consumption [39]. Under the conditions of the 36-MHz system clock, the 1.2-V internal voltage, and the 3.3-V IO voltage, the total power consumption of this chip is 23 mW. This chip is fabricated using SMIC CMOS 110-nm process and packaged in QFN56. The co-design area of SM2, SM3, and SM4 cryptographic module is about 1.0 mm^2. Fig. 10 shows the ASIC chip layout and Table VIII gives the resource occupancy.

In order to highlight the advantages of SW/HW co-design methods in terms of resource utilization and area costs, Table VIII compares the scheme of SW/HW co-design with the pure hardware design of SM3 and SM4. As can be seen from Table VIII, the number of gates or area costs of SM3 in pure hardware design is increased 6.29% than SW/HW co-design, and the gates or area of SM4 is increased 8.73%. Therefore, compared with the pure hardware implementation, SW/HW co-design can save multiple register resources and area. In addition, SW/HW co-design implementation is more flexible and easier to change the algorithm than pure hardware implementation.

V. CONCLUSION

Based on resources usage and run-time analysis of a software-only implementation of given encryption/decryption and digital signature system, functionality of the system is rationally partitioned into hardware and software modules. A typical algorithm flow (SHT) is proposed and implemented using software/hardware co-design approach. This flow implementation achieved roughly 35 times speed-up when compared with software-only implementation. An improved hybrid algorithm flow (SHHI) is then investigated and implemented using parallelism, algorithm improvement, and performance optimization. The overall performance of SHHI is 12.35% higher than that of SHT, which is 96.19% higher than software-only method.

In the future, the construction method and optimization of Sbox in SM4 will be studied to further reduce the implementation complexity of SM4. The reconfigurability and portability of the hardware modules will be further enhanced to extend the applications of the system and modules to resist side-channel and other attacks will be added to increase the security of the system.

ACKNOWLEDGMENT

The authors would like to thank Dr. Y. Liu for his help during the writing of this article.

REFERENCES

[1] J. Kim, H. Wu, and R. C.-W. Phan, "Cryptography and future security," *Discr. Appl. Math.*, vol. 241, p. 1, May 2018.

[2] *Announcing the Advanced Encryption Standard (AES)*, FIPS Standard 197, pp. 1–51, 2001.

[3] W. N. Chelton and M. Benaissa, "Fast elliptic curve cryptography on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 16, no. 2, pp. 198–205, Feb. 2008.

[4] A. W. Appel, "Verification of a cryptographic primitive: SHA-256," *ACM Trans. Program. Lang. Syst.*, vol. 37, no. 2, p. 7, 2015.

[5] *IT Security Techniques-Digital Signatures With Appendix—Part 3: Discrete Logarithm Based Mechanisms*, ISO/IEC Standard 14888-3, Nov. 2018.

[6] *IT Security Techniques-Hash Functions–Part 3: Dedicated Hash Functions*, ISO/IEC Standard 10118-3, Oct. 2018.

[7] *Information Technology-Security Techniques-Encryption Algorithms—Part 3: Block Ciphers*, ISO/IEC Standard 18033-3, Dec. 2010.

[8] A. Yang, J. Nam, M. Kim, and K.-K. R. Choo, "Provably-secure (Chinese Government) SM2 and simplified SM2 key exchange protocols," *Sci. World J.*, vol. 2014, Sep. 2014, Art. no. 825984.

[9] Z. Zhang, K. Yang, J. Zhang, and C. Chen, "Security of the SM2 signature scheme against generalized key substitution attacks," in *Proc. Int. Conf. Res. Security Standardisation*, 2015, pp. 140–153.

[10] D. Bai, H. Yu, G. Wang, and X. Wang, "Improved boomerang attacks on round-reduced SM3 and keyed permutation of BLAKE-256," *IET Inf. Security*, vol. 9, no. 3, pp. 167–178, May 2014.

[11] S. Pu *et al.*, "Boolean matrix masking for SM4 block cipher algorithm," in *Proc. IEEE 13th Int. Conf. Comput. Intell. Security (CIS)*, 2017, pp. 238–242.

[12] D. Wang, L. Wu, and X. Zhang, "Key-leakage hardware trojan with super concealment based on the fault injection for block cipher of SM4," *Electron. Lett.*, vol. 54, no. 13, pp. 810–812, Jun. 2018.

[13] M. Knežević, V. Nikov, and P. Rombouts, "Low-latency ECDSA signature verification—A road toward safer traffic," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 11, pp. 3257–3267, Nov. 2016.

[14] J. Haase, T. Oberthür, and M. Oberwestberg, "Design methodology for IP providers," in *Reuse Techniques for VLSI Design*. Boston, MA, USA: Springer, 1999, pp. 49–62.

[15] T. Sledevič and R. Laptik, "An evaluation of hardware-software design for sound source localization based on SoC," in *Proc. IEEE Open Conf. Elect. Electron. Inf. Sci. (eStream)*, 2017, pp. 1–4.

[16] B. Wu, C. Peng, and X. Sun, "Design of rapid prototyping platform for hardware/software co-design of embedded system," *J. Comput.-Aided Design Comput. Graph.*, vol. 7, p. 1, Jul. 2003.

[17] J. Kokila, N. Ramasubramanian, and S. Indrajeet, "A survey of hardware and software co-design issues for system on chip design," in *Advanced Computing and Communication Technologies*. Singapore: Springer, 2016, pp. 41–49.

[18] L. A. Tambara *et al.*, "Reliability–performance analysis of hardware and software co-designs in SRAM-based APSoCs," *IEEE Trans. Nucl. Sci.*, vol. 65, no. 8, pp. 1935–1942, Aug. 2018.

[19] N. K. Jayakodi, A. Chatterjee, W. Choi, J. R. Doppa, and P. P. Pande, "Trading-off accuracy and energy of deep inference on embedded systems: A co-design approach," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2881–2893, Nov. 2018.

[20] V. Migliore, M. M. Real, V. Lapotre, A. Tisserand, C. Fontaine, and G. Gogniat, "Hardware/software co-design of an accelerator for FV homomorphic encryption scheme using Karatsuba algorithm," *IEEE Trans. Comput.*, vol. 67, no. 3, pp. 335–347, Mar. 2018.

[21] W. M. Lim and M. Benaissa, "Design space exploration of a hardware-software co-designed GF(2*m*) Galois field processor for forward error correction and cryptography," in *Proc. IEEE/ACM/IFIP Int. Conf. Hardw. Softw. Codesign Syst. Synth.*, 2003, pp. 53–58.

[22] A. Hafsa, N. Alimi, A. Sghaier, M. Zeghid, and M. Machhout, "A hardware-software co-designed AES-ECC cryptosystem," in *Proc. IEEE Int. Conf. Adv. Syst. Elect. Technol. (IC_ASET)*, 2017, pp. 50–54.

[23] M. U. Sharif, R. Shahid, K. Gaj, and M. Rogawski, "Hardware-software codesign of RSA for optimal performance vs. flexibility trade-off," in *Proc. IEEE 26th Int. Conf. Field Program. Logic Appl. (FPL)*, 2016, pp. 1–4.

[24] C. Xiao, L. Zhang, Y. Xie, W. Liu, and D. Liu, "Hardware/software adaptive cryptographic acceleration for big data processing," *Security Commun. Netw.*, vol. 2018, Aug. 2018, Art. no. 7631342.

[25] M. M. A. Basiri and S. K. Shukla, "Efficient hardware-software codesigns of AES encryptor and RS-BCH encoder," in *Proc. Int. Symp. VLSI Design Test*, 2018, pp. 3–15.

[26] T. Feten, K. Halim, and L. Younes, "Hardware/software co-design using high level synthesis for cryptographic module," in *Proc. IEEE 7th Int. Conf. Model. Identification Control (ICMIC)*, 2015, pp. 1–6.

[27] P. Dixit, J. Zalke, and S. Admane, "Speed optimization of AES algorithm with hardware-software co-design," in *Proc. IEEE 2nd Int. Conf. Convergence Technol. (I2CT)*, 2017, pp. 793–798.

[28] I. Vasyltsov, E. Hambardzumyan, Y.-S. Kim, and B. Karpinskyy, "Fast digital TRNG based on metastable ring oscillator," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.*, 2008, pp. 164–180.

[29] M. Scott. (2003). *Multiprecision Integer and Rational Arithmetic C/C++ Library (MIRACL)*. [Online]. Available: http://www.shamus.ie

[30] X. Du and S. Li, "The ASIC implementation of SM3 hash algorithm for high throughput," *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, vol. 99, no. 7, pp. 1481–1487, 2016.

[31] Y. Godhal, K. Chatterjee, and T. A. Henzinger, "Synthesis of AMBA AHB from formal specification: A case study," *Int. J. Softw. Tools Technol. Transfer*, vol. 15, nos. 5–6, pp. 585–601, 2013.

[32] M. Kaihara and N. Takagi, "Bipartite modular multiplication method," *IEEE Trans. Comput.*, vol. 57, no. 2, pp. 157–164, Feb. 2008.

[33] S. Ghosh, M. Alam, I. S. Gupta, and D. R. Chowdhury, "A robust GF(p) parallel arithmetic unit for public key cryptography," in *Proc. IEEE 10th Euromicro Conf. Digit. Syst. Design Architect. Methods Tools (DSD)*, 2007, pp. 109–115.

[34] J. M. Hammersley and D. C. Handscomb, "Random, pseudorandom, and quasirandom numbers," in *Monte Carlo Methods*. Dordrecht, The Netherlands: Springer, 1964, pp. 25–42.

[35] M. M. Deshpande and M. A. Hasamnis, "Design of encryption system using NIOS II processor," *Int. J. Comput. Appl.*, vol. 68, no. 21, pp. 8–13, 2013.

[36] Y. Liu, W. Guo, Y. Tan, J. Wei, and D. Sun, "An efficient scheme for implementation of SM2 digital signature over GF(p)," in *Contemporary Research on e-Business Technology and Strategy*. Heidelberg, Germany: Springer, 2012, pp. 250–258.

[37] H. Wang and Q. Li, "Efficient implementation of public key cryptosystems on mote sensors (short paper)," in *Proc. Int. Conf. Inf. Commun. Security*, 2006, pp. 519–528.

[38] B. Panjwani and D. C. Mehta, "Hardware-software co-design of elliptic curve digital signature algorithm over binary fields," in *Proc. IEEE Int. Conf. Adv. Comput. Commun. Informat. (ICACCI)*, 2015, pp. 1101–1106.

[39] X. Kavousianos and K. Chakrabarty, "Testing for SoCs with advanced static and dynamic power-management capabilities," in *Proc. Conf. Design Autom. Test Europe*, 2013, pp. 737–742.

Xin Zheng received the B.S. degree in automation from the Huazhong University of Science and Technology Wuchang Branch, Wuhan, China, in 2014, and the M.S. degree in control engineering from the Guangdong University of Technology, Guangzhou, China, in 2018, where she is currently pursuing the Ph.D. degree with the Department of Automation.

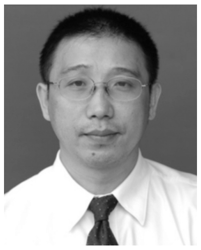
Her current research interests include software and hardware co-design, information security, and machine learning for graph data.

Chongyao Xu received the B.S. degree in electronic information engineering from Zhoukou Normal University, Zhoukou, China, in 2014, and the M.S. degree in control engineering from the Guangdong University of Technology, Guangzhou, China, in 2018. He is currently pursuing the Ph.D. degree in electronics and computer engineering with the University of Macau, Macau, China.

His current research interest includes physical unclonable function.

Xianghong Hu received the B.S. degree in communication engineering from the Guangdong University of Technology, Guangdong, China, in 2015, where he is currently pursuing the Ph.D. degree with the Department of Automation.

His current research interests include computer architecture, deep learning, information security, and very large-scale integration design.



Yun Zhang received the B.S. and M.S. degrees in automatic engineering from Hunan University, Changsha, China, in 1982 and 1986, respectively, and the Ph.D. degree in automatic engineering from the South China University of Science and Technology, Guangzhou, China, in 1998.

He is currently a Full Professor with the School of Automation, Guangdong University of Technology, Guangzhou. His current research interests include intelligent control systems, multiagent systems, neural networks, and signal processing.

Xiaoming Xiong received the B.S. degree from the Department of Radio, South China University of Technology, Guangzhou, China, in 1982, and the Ph.D. degree from the Department of Electrical Engineering and Communication, University of California at Berkeley, Berkeley, CA, USA, in 1988.

He is currently a Professor with the Guangdong University of Technology, Guangzhou. He is also the Chief Technical Advisor of Chipeye Microelectronics Foshan Ltd., Foshan, China. His current research interests include hardware/software cryptographic, system-on-chip design, very large-scale integration design, electronic design automation, computer algorithms, and graph theory.