On the Design and Performance of Chinese OSCCA-approved Cryptographic Algorithms

Louise Bergman Martinkauppi*, Qiuping He† and Dragos Ilie‡
Dept. of Computer Science
Blekinge Institute of Technology (BTH), Karlskrona, Sweden
*louisemartinkauppi@hotmail.com, †ping95_@hotmail.com, ‡dragos.ilie@bth.se

***Abstract*—SM2, SM3, and SM4 are cryptographic standards authorized to be used in China. To comply with Chinese cryptography laws, standard cryptographic algorithms in products targeting the Chinese market may need to be replaced with the algorithms mentioned above. It is important to know beforehand if the replaced algorithms impact performance. Bad performance may degrade user experience and increase future system costs.**

We present a performance study of the standard cryptographic algorithms (RSA, ECDSA, SHA-256, and AES-128) and corresponding Chinese cryptographic algorithms.

Our results indicate that the digital signature algorithms SM2 and ECDSA have similar design and also similar performance. SM2 and RSA have fundamentally different designs. SM2 performs better than RSA when generating keys and signatures. Hash algorithms SM3 and SHA-256 have many design similarities, but SHA-256 performs slightly better than SM3. AES-128 and SM4 share some similarities in the design. In the controlled experiment, AES-128 outperforms SM4 with a significant margin.

I. Introduction

The Chinese IT industry has grown dramatically over the past two decades and the value of e-commerce in China accounts for more than 40% of all worldwide e-commerce today [1]. If a company wants to establish itself in China, they need to follow Chinese law. Legislation concerning the use of cryptography is defined by the State Cryptography Administration (SCA). Commercial encryption is handled by the Office of State Commercial Cryptography Administration (OSCCA), which is managed by SCA.

Until recently, China was regulating the import, export, sale, use, and research of commercial encryption based on legislation issued in 1999 [2]. The regulations prohibited foreign encryption products and required all companies or individuals selling or producing commercial encryption products to gain the OSCCA's approval. The OSCCA-published algorithms SM2, SM3, SM4, SM9, and ZUC are commercial cryptographic algorithms mandated by the SCA to be used within China [3].

In 2019, China has adopted the Cryptography Law, which regulates the use of encryption methods[1] to protect digital data in transit and at rest [4]. The law, which came into effect on January 1, 2020, divides encryption into three different categories: core, ordinary and commercial.

Core and ordinary encryption are used for protecting China's state secrets at different classification levels. Both core and ordinary encryption are considered state secrets and thus are strictly regulated by the SCA. It is therefore likely that these types of encryption will be based on the national algorithms mentioned above, so that SCA can exercise full control over standards and implementations.

Commercial encryption is used to protect information that is not considered a state secret. The new law removes many restrictions from the 1999 regulations. In particular, it allows use of foreign commercial encryption products that have completed a certification process. The goal is to verify that the products comply with Chinese national standards and regulations. Critical infrastructure operators who use commercial encryption products are required to perform security assessments on their use of encryption, and in some cases even undergo national security reviews. Unfortunately, currently there is no clear definition of what technologies fall under ”commercial encryption”. Also, the details of the certification process are not available yet [5].

Given the current state, it may be easier for a company to modify their products for the Chinese market to use the OSCCA-approved encryption algorithms. The problem is that the stakeholders do not know how the replacing algorithms will affect the performance of their products. They may increase response times, energy consumption, and memory utilization, which may increase capital and operational expenses.

In this paper, we provide an overview of the design of Chinese OSCCA-approved algorithms. In addition, we quantify the performance impact, in terms of execution time and memory usage, when replacing standard cryptographic algorithms with corresponding Chinese cryptographic algorithms.

The reminder of this paper is organized as follows. Section II compares the design of standard cryptographic algorithms with Chinese counterparts. Related work is presented in Section III. In Section IV we describe the testbed and the experiments used for investigating the algorithm performance. The analysis of the results from the experiment is presented in Section V and Section VI shares our concluding remarks.

‡Corresponding author.

[1] The law does not address specifically decryption of encrypted data or key management.

©2020 IEEE. Published paper is available on IEEE Xplore Digital Library
https://doi.org/10.1109/COMM48946.2020.9142035

TABLE I
SM2, ECDSA, AND RSA PROPERTIES.

	ECDSA	**SM2**	**RSA**
Type	Asymmetric key algorithm	Asymmetric cryptosystem	Asymmetric cryptosystem
Based on	Elliptic curve discrete logarithm problem	Elliptic curve discrete logarithm problem	Integer factorization problem
Used for	Digital signatures	Digital signatures Encryption & decryption Key exchange	Digital signatures Encryption & decryption Key exchange
Public key	$Q = d \times G$	$P = d \times G$	$\langle N, e\rangle$, $N = p \cdot q$
Private key	d (random integer)	d (random integer)	$\langle N, d\rangle$, $N = p \cdot q$
Recommended key length (bits)	**P-256** Private key: 256 Public key: 512 **P-384** Private key: 384 Public key: 768	**SM2 curve** Private key: 256 Public key: 512	2048
Digital signature auxiliary functions	Hash function (SHA-1 or SHA-2) Random number generator	Hash function (SM3) Random number generator	PSS Prime number generator

TABLE II
SM3 AND SHA-256 PROPERTIES.

	SM3	**SHA-256**
Structure	Merkle-Damgård	Merkle-Damgård
Compression function	Davies-Meyer	Davies-Meyer (based on)
Input (bits)	$0 \leq l \leq 2^{64}$	$0 \leq l \leq 2^{64}$
Output (bits)	256	256
Rounds	64	64
Operations	ADD, XOR, NOT, OR, ADD (mod 2^{32}), Concatenation, ROTL	ADD, XOR, NOT, ADD (mod 2^{32}) , SHR, Concatenation, ROTR
Constants (words)	2	64

TABLE III
SM4 AND AES-128 PROPERTIES.

	SM4	**AES-128**
Type	Block cipher	Block cipher
Structure	Unbalanced Feistel Network (UFN)	Substitution–permutation network (SPN)
Field(s)	$GF(2^8)$ and $GF(2)$	$GF(2^8)$ and $GF(2)$
Block size (bits)	128	128
Key length (bits)	128	128
Round keys	32 keys á 32 bits	11 keys á 128 bits
Number of rounds	32	10
S-box	Inversion-based mapping	Inversion-based mapping
Number of S-box lookups	128	160
Operations	XOR, Sbox, cyclic bit shifts	XOR, Sbox, cyclic bit shifts, modular multiplication

II. OVERVIEW OF CRYPTOGRAPHIC ALGORITHMS

This section presents a comparison of the design of the algorithms included in this study. The algorithms are grouped into public-key schemes, hashing algorithms, and algorithms for symmetric encryption. A more detailed description of the characteristics of these algorithms can be found in [6].

A. Public-key schemes: SM2, ECDSA, and RSA

SM2, ECDSA, and RSA are all based on asymmetric encryptions. The ECDSA and SM2 public key, Q and P, are points on a selected elliptic curve, while RSA's public key is a pair of two positive integers $\langle N, e\rangle$. In ECDSA and SM2, the private key is a random integer which satisfies either $d \in [1, n-1]$ (ECDSA) or $d \in [1, n-2]$ (SM2). The RSA private key is a pair of positive integers $\langle N, d\rangle$. The design details of these algorithms are summarized in Table I.

The RSA decryption and signature generation operations tend to be computationally heavy because both d and N are large integers. The workloads for encryption/signature verification and decryption/signature generation are therefore often unbalanced in RSA. Although RSA is expected to perform worse than elliptic curve cryptography (ECC) based algorithms, we included it in our study for two reasons. The first one is that it provides a baseline from which to measure performance gains when using ECC. The second reason is that ECC performance results similar to or lower than those for RSA would indicate an implementation issue.

The ECC key generation consists of choosing a point G on the elliptic curve, generating a random integer d, and calculating $P = d \times G$, where $\times$ is a point multiplication operator defined specifically for ECC. The point multiplication operation has higher time complexity than other operation in ECC [7], requiring the largest execution time [8]. Since the signature verification for both ECDSA and SM2 performs two point multiplications, compared with one point multiplication in the signature generation, the verification is slower.

B. Hashing algorithms: SM3 and SHA-256

Both algorithms take an input M with length l, where $0 \leq l \leq 2^{64}$, as shown in Table II. The message is padded using Merkle-Damgård construction. It is then divided into blocks of the same block length. These blocks are processed one at a time by a compression function, where the output serves as input to the compression of the next block. The compression function operates 64 rounds for each message blocks and outputs a 256 bits message digest.

SM3 uses a Davies-Meyer compression function while SHA-256 uses a compression function *similar* to Davies-Meyer. The difference lies in how the chaining value is combined with the block cipher output to create the next chaining value. In SM3 an XOR operation is used, and in SHA-256 an ADD operation is used [6].

The compression function of SM3 is more complicated than compression functions in other common hash algorithms. The SM3 compression function limits the algorithm's throughput, because of the rotational shift, ADD, and XOR operations which leads to a long circuit delay [9].

C. Symmetric encryption: SM4 and AES-128

As shown in Table III, SM4 is based on a Unbalanced Feistel Network (UFN) while AES is based on a Substitution-Permutation Network (SPN). In general, SPNs have more built-in parallelism [10]. AES implementations that can exploit this property can be potentially faster than SM4, especially if one considers that AES requires fewer rounds.

SM4 has identical encryption and decryption algorithms. Only the order of the round keys has to be reversed for decryption. For AES, the differences in the encryption and decryption processes require the encryption and decryption algorithms to be implemented separately.

Most modern Intel processors support Intel Advanced Encryption Standard (AES) New Instructions (AES-NI). These instructions are designed to accelerate the performance of the AES algorithm by 3–10 times over a complete software implementation [11]. This type of acceleration is not available for SM4. However, there are scenarios (*e. g.*, embedded devices, IoT) where other processors are used and AES cannot benefit from acceleration capabilities. Therefore, when comparing the performance of SM4 and AES-128, we look at both cases (with and without AES-NI instructions).

III. Related Work

Due to space restritions, we only focus on work related to the Chinese encryption algorithms.

Bai *et al.* [12] did a theoretical comparative study of SM2 and the international standard ECC algorithm (which seems to refer to ECDSA, although this is not clearly mentioned), with regards to efficiency and security. They concluded that the SM2 algorithm is better than the international standard ECC algorithm, because the SM2 signature covers ECC parameters that can be used to verify the correctness of the plaintext. However, no practical experiments were performed.

Feng *et al.* [13] designed and implemented a testing and evaluation system for trusted computing (TC) platforms. As part of this work they tested the correctness and performance of cryptographic algorithms used inside the secure chips and trusted software stacks. Their test cases included RSA, SM2, SM3, SM4, and HMAC. The authors conclude that SM2 is faster than RSA. However, no data is presented, and the reproducibility is limited.

Bai and Zhao [14] investigated the speed limit of SM2 point multiplication in serial and parallel architectures without considering any resource constraints. The authors also compared the performance difference between non-adjacent form (NAF) and w-NAF encoding to get more detailed information about the SM2 speed limit. Thus, the results of this study apply only to a subpart of the SM2 algorithm.

Zhao and Bai [8] presented a high-performance point multiplication scheme for SM2. This operation was optimized through the use of a one-cycle full-precision multiplier. During hardware evaluation, their SM2 architecture performed over 49000 point multiplications per seconds, which was the highest known single core performance, according to the authors.

Mendel *et al.* [15] concluded that the design of SM3 is very similar to SHA-256, extended the methods for collision attacks on SHA-256 and applied them to SM3. They produced two collision attacks on round-reduced SM3 with practical complexity. The authors managed to construct collisions for up to 20 rounds of SM3. Free-start collisions (collisions where the attacker decides the initialisation vector IV) were constructed for up to 24 rounds of SM3.

Ao *et al.* [16] presented a compact hardware implementation of SM3 using SRAM instead of shift registers for the message expansion function. The authors compared the throughput of their hardware implementation with a SHA-256 and SM3 hardware implementation from other studies. The compact architecture which they implemented can be used in a resource-constrained system due to its low-cost and low-power consumption.

Ma *et al.* [17] evaluated and optimized the hardware performance of the hash algorithm SM3 on Field-Programmable Gate Array (FPGA) circuits. They proposed new optimization techniques for SM3 since existing optimization techniques are not applicable to SM3. Their study focuses on the hardware performance of FPGA and the optimization techniques proposed by Ma et al. can be used for the implementation of other hash algorithms, such as SHA-2.

Liu *et al.* [18] analyzed the SM4 algorithm and investigated the origin of the SM4 S-box. They concluded that the design of SM4 is influenced by Rijndael (*i. e.*, AES), mostly because both algorithms use an inversion-based mapping. Although the major focus of the paper is the origin of the SM4 S-box, the authors are indirectly comparing SM4 with AES.

Cheng and Ding [19] described the block ciphers DES, AES, and SM4, as well as the overall design theory and the structure of block ciphers. They also reviewed common cryptoanalysis methods related to block ciphers. However, no outright comparisons of the block cipher algorithms are described in the paper.

Ji *et al.* [20] examined algebraic attacks against SM4 and compared the resistances of SM4 with that of AES. Their result indicate that SM4 seems stronger than AES.

Li *et al.* [21] implemented a new encryption and authentication scheme SM4-GCM on FPGA circuits. The authors proved that the new design of encryption and authentication scheme has a higher throughput and lower resource consumption than AES-GCM by conducting a hardware performance evaluation.

IV. Experimental Setup

The aim of our experiments is to compare the performance of the standard cryptographic algorithms to the performance of corresponding Chinese algorithms in the following applications:

Experiment 1: key generation, digital signature, signature verification (SM2 *vs.* ECDSA *vs.* RSA)
Experiment 2: hashing (SM3 *vs.* SHA-256)
Experiment 3: symmetric encryption and decryption (SM4 *vs.* AES-128 *vs.* AES-128-NI).

For symmetric encryption and decryption we considered the following modes: Electronic Code Book (ECB), Cipher Block Chaining (CBC), and Counter (CTR) [22]. Although ECB is considered unsafe, it can still be used as a baseline for benchmarking. Due to time constraints we have not considered more advanced methods, such as authenticated encryption.

We have used the following cryptographic libraries for our experiments: OpenSSL v1.1.1b[2], GmSSL v2.5.0[3], and Botan v2.11.0[4]. Originally, the Chinese algorithms where implemented in GmSSL, which is a fork of OpenSSL v1.1.0d. They were later implemented in OpenSSL.

We strived to use the most recent algorithm implementations in order to benefit from any potential bug fixes and code optimizations. However, we replaced OpenSSL with GmSSL in the digital signature experiments. The reason is that the

[2] https://www.openssl.org

[3] http://gmssl.org

[4] https://botan.randombit.net

command-line tools from OpenSSL at the time when the experiment was performed lacked the possibility to specify which signature algorithm to use (only ECDSA was supported). Botan is a C++ library that is linked to a C++ program we wrote. The program implements all functionality required to execute the tests we used for OpenSSL/GmSSL. However, our program has less overhead than OpenSSL/GmSSL command-line tool because it only implements the bare necessary functionality for the tests. Therefore a comparison between the two is not fair. The aim is instead to explore if the relative performance of the algorithms (standard *vs.* Chinese) is the same under both frameworks.

During the experiments we collected the following performance metrics for the executing tool: elapsed real-time, CPU time, CPU cycles, and resident set size (RSS). Real-time is actual wall clock time required for the execution of the process, including waiting for any I/O activities to complete. CPU time is time spent in executing on one or multiple CPUs without taking I/O into account. This can also be expressed as CPU cycles, which makes the values easier to compare between CPUs with different clock rates. RSS shows how much memory a process is currently using in RAM.

Real-time and RSS measurements were collected using the command `time(1)` [23]. The command `perf-stat(1)` was used for collecting CPU time and cycles [24].

We have created a file of size 1 GB with random content from `/dev/surandom` to be used for our experiments. This file and any additional files required or created by the three experiments were placed on a ramdisk created with the RAM block disk device driver in Linux. The purpose of the ramdisk was to alleviate the effects of disk I/O on the measurements.

For each experiment we used a sample size of 100 measurements which strikes a reasonable balance between sample size and experiment duration. All experiments were replicated 120 times and the first 20 samples were thrown away, to ensure fair cache advantages for each experiment. For each sample, we have computed the following statistics: minimum, maximum, mean, median, and standard deviation.

The experiments were carried out on a host equipped with Intel Core i7-8650U CPU, 32 GB RAM and Ubuntu 16.04 LTS 64-bit OS. All scripts and C++ programs developed for this project are available from our GitHub repository[5]. The commands were all executed at the highest scheduling priority in user space in order to alleviate the effect of other processes competing for the CPU.

V. Peformance Results and Analysis

The following sections present the results from the experiments described in Section IV. The bar graphs in every section present the mean of the metrics of each algorithm. The bar whiskers indicate the standard deviation.

In general, our results show that the real-time measurements match almost identically the CPU time and cycles measurements. Most likely, this happens because the ramdisk diminishes the effects of disk I/O. Furthermore, memory usage *in all cases* was in the range 4–5 MB. Due to space constraints, we have chosen to show only results for real-time CPU usage. However, a complete record of all statistics is available in [6].

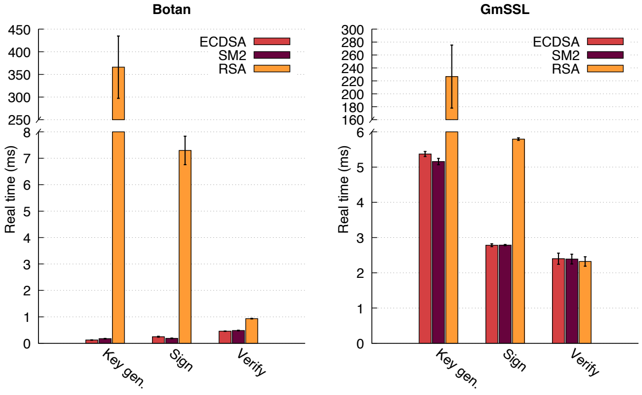

Fig. 1. Digital signature real-time in Botan and GmSSL.

A. Experiment 1: Digital Signature Algorithms

Fig. 1 shows the real-time CPU usage required to generate a key, sign a 256-bit message and verify the signature for ECDSA, SM2, and RSA. It is important to mention that the results for signing or verifying a message include the time required to compute a 256-bit hash values over the message (SHA-256 for ECDSA and RSA, SM3 for SM2).

Key generation and signing operations for RSA takes substantially longer time than for SM2 and ECDSA, both in Botan and GmSSL. More specifically, in Botan SM2 key generation required approximately 40% longer time than ECDSA key generation. Further, in GmSSL, signing with RSA takes approximately double the time compared to signing with SM2 and ECDSA, while in Botan the difference is 30–40 times larger. Verification with RSA takes almost twice as long time than verification with SM2 in Botan, while in GmSSL the RSA verification is slightly faster than that of SM2. Apart from small deviations, SM2 and ECDSA show similar performance both in Botan and GmSSL.

The key generation of RSA is substantially slower than SM2 because of the operations required to generate a key is more complex. RSA key generation requires two large prime integers to be generated. This results in very dispersed time values since several integers may have to go through the primality tests before a integer passes and is considered a prime. Working with large integers also adds to the time taken for RSA key generation. SM2 and ECDSA key generation is much faster since the only operations required is one random number generation and one point multiplication.

The discovery that the signing operation is slower for RSA than for SM2 and ECDSA corroborate results from other studies [25], [26] comparing RSA with ECDSA. The two studies also found that signature verification with RSA is faster than with ECDSA. However, our results on this matter are not consistent as RSA verification is slower than SM2 in Botan, but faster in GmSSL.

We inspected the Botan source code in search of causes for the 40% longer time for SM2 key generation. We discovered

[5] https://github.com/qiupinghe/BTH-TE2502-MasterThesis

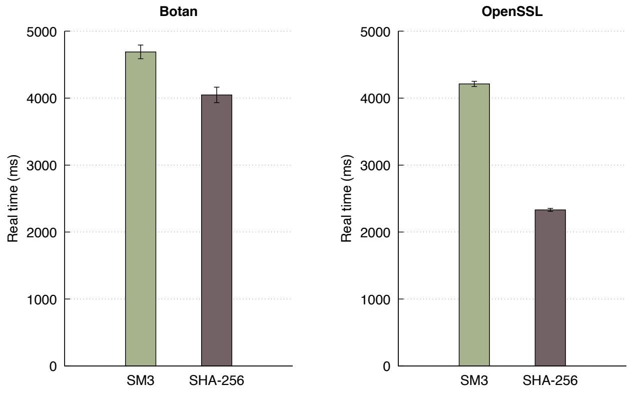

Fig. 2. Hash algorithms real-time in Botan and OpenSSL.

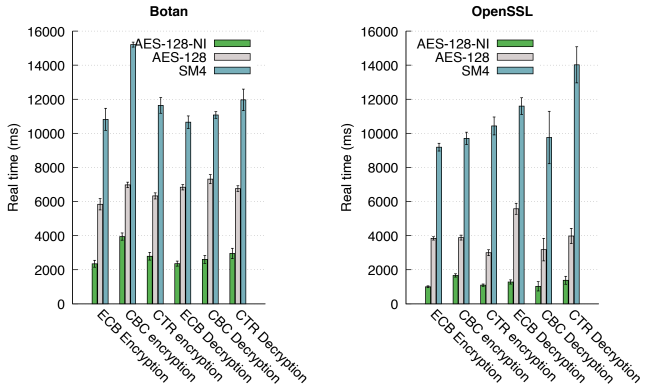

Fig. 3. Block Ciphers real-time graphs in Botan and OpenSSL.

that during SM2 key generation, Botan computes another value that is used later during signing. We think these computations may account for the key generation performance. The precomputed value may also be the reason why in Botan SM2 signing is faster than ECDSA signing. Unfortunately, due to time constraints we have not followed this avenue any further.

B. Experiment 2: Hashing Algorithms

In Fig. 2 it can be seen that the CPU real-time usage required to hash a 1GB file is greater for SM3 than SHA-256 in both Botan and OpenSSL. In OpenSSL, the real-time taken to hash the file using SM3 is twice as large than for SHA-256, while in Botan the difference is smaller.

The experiment result showed that SM3 is 81% slower than SHA-256 in OpenSSL, while SM3 is 16% slower than SHA-256 in Botan. As described in Section II-B, SHA-256 and SM3 have a very similar design. However, SM3 requires 128 additional assignment operations compared to SHA-256 [6]. This characteristic may have a tiny effect when hashing small files, but for larger files, it could bring a significant impact on the performance, since there are 128 more assignment instructions per 64 bytes of data.

In addition, SHA-256 is a more mature hashing algorithm that OpenSSL has been supporting since version *0.9.8o* from 2010 [27]. OpenSSL has added support for SM3 in version *1.1.1b* in 2018 [28]. Therefore, the OpenSSL code for SHA-256 is probably better optimized for performance.

C. Experiment 3: Symmetric Encryption Algorithms

The results from all block cipher tests showed that SM4 is much slower than AES-128 and AES-128-NI regardless of operation or mode.

Figure 3 shows that AES-128-NI is the fastest algorithm in Botan, much thanks to hardware acceleration from the CPU. Even without hardware acceleration, AES-128 is still faster than SM4. Botan ECB and CTR mode show similar performance, while CBC mode is consistently slower.

The AES algorithms are faster than SM4 in the OpenSSL tests as well. Rather surprisingly, CBC mode outperforms the other modes for decryption.

In our experiments, the observed performance differences do not depend on the parallelism ability inherent in SPNs, because all block cipher tests ran on a single CPU core. However, because the block cipher tests were executed on a processor with x86 64-bit architecture, this may have worked towards AES's advantage. AES may have been able to use registers more optimally since the round key size of AES is 128 bits long, while round keys of SM4 are 32 bits long.

VI. Concluding Remarks

The aim of this study was to provide a quantitative measure of the performance gain or loss when replacing standard cryptographic algorithms with Chinese counterparts. To do so, we have computed a relative statistic related to mean CPU real-time usage, for each metric presented in Section V. Given a metric M, the statistic is computed as $(M_c - M_s) * 100/M_s$ where M_s and M_c are the metric values (*i.e.*, mean CPU real-time) for the standard algorithm and Chinese algorithm, respectively. The results are shown in Table IV.

The results are marked with three colours, red, yellow, and green, that have the following meaning:

- Red: Relative percentage change is higher than +20%. Fields marked with red mean that replacement of the standard algorithm will impact the performance negatively.
- Yellow: Relative percentage change is between -20% — +20%. Fields marked with yellow mean that the replacement with SM algorithms will cause a relatively small performance impact.
- Green: Relative percentage change lower than -20%. Fields marked with green mean that the replacement of the standard algorithm will improve the performance.

A rather blunt observation is that although the Chinese algorithms perform better or equally well for digital signature operations, their performance during symmetric encryption operations is significantly worse with performance hits in the range the 85 – 915%, compared to standard algorithms. Whether this type of performance loss (related specifically to CPU real-time usage) for symmetric encryption is acceptable, depends on the specific type of application considered.

We believe the results presented here are valuable to anyone who wants to establish a product in China or integrate their

TABLE IV

RELATIVE PERFORMANCE OF ALGORITHMS IN THE STUDY.

Library	Operation	Mode	Algorithm 1	Algorithm 2	Percentage Change(%)
Digital Signature Algorithms					
GmSSL	Key	-	RSA	SM2	-97,72
	Sign	-	RSA	SM2	-51,92
	Verify	-	RSA	SM2	2,93
	Key	-	ECDSA	SM2	-4,00
	Sign	-	ECDSA	SM2	0,10
	Verify	-	ECDSA	SM2	-0,40
Botan	Key	-	RSA	SM2	-99,95
	Sign	-	RSA	SM2	-97,45
	Verify	-	RSA	SM2	-48,22
	Key	-	ECDSA	SM2	40,30
	Sign	-	ECDSA	SM2	-24,69
	Verify	-	ECDSA	SM2	5,47
Hash Algorithms					
OpenSSL	-	-	SHA-256	SM3	80,68
Botan	-	-	SHA-256	SM3	15,85
Block Cipher Algorithms					
OpenSSL	Encryption	ECB	AES-128	SM4	139,30
		CBC	AES-128	SM4	149,89
		CTR	AES-128	SM4	247,84
		ECB	AES-128-NI	SM4	817,69
		CBC	AES-128-NI	SM4	483,01
		CTR	AES-128-NI	SM4	852,69
	Decryption	ECB	AES-128	SM4	108,09
		CBC	AES-128	SM4	206,86
		CTR	AES-128	SM4	252,39
		ECB	AES-128-NI	SM4	805,09
		CBC	AES-128-NI	SM4	849,97
		CTR	AES-128-NI	SM4	915,61
Botan	Encryption	ECB	AES-128	SM4	85,29
		CBC	AES-128	SM4	117,99
		CTR	AES-128	SM4	83,85
		ECB	AES-128-NI	SM4	360,41
		CBC	AES-128-NI	SM4	285,57
		CTR	AES-128-NI	SM4	316,94
	Decryption	ECB	AES-128	SM4	55,81
		CBC	AES-128	SM4	51,39
		CTR	AES-128	SM4	77,05
		ECB	AES-128-NI	SM4	351,86
		CBC	AES-128-NI	SM4	325,36
		CTR	AES-128-NI	SM4	304,44

product with a Chinese system since it can be necessary to replace an existing cryptographic algorithm with a Chinese correspondence. Organizations looking to replace standard algorithms with Chinese algorithms will need to perform additional analysis studies specific to their products and make an informed decision.

We would have liked to provide a comparison with SHA-3, the newer hashing algorithm gaining support with many applications. However, the GmSSL library that was used for our tests did not have SHA-3 support at the time the study was carried out. Thus, we decided to postpone this comparison for future work.

REFERENCES

[1] K. W. Wang, J. Woetzel, J. Seong, J. Manyika, M. Chui, and W. Wong. (2017, Dec.) Digital China: Powering the economy to global competitiveness. [Online]. Available: https://www.mckinsey.com/featured-insights/china/digital-china-powering-the-economy-to-global-competitiveness

[2] Covington. (2017, May) China releases draft encryption law for public comment. [Online]. Available: https://www.cov.com/en/news-and-insights/insights/2017/05/china-releases-draft-encryption-law-for-public-comment

[3] D. Li and Y. Liu, "Introduction to the commercial cryptography scheme in china," https://icmconference.org/wp-content/uploads/C03b-Li.pptx.pdf, May 2016.

[4] L. Dong. (2019, Oct.) Cryptography law of the people's republic of china. [Online]. Available: http://www.npc.gov.cn/npc/c30834/201910/6f7be7dd5ae5459a8de8baf36296bc74.shtml

[5] S. Dickinson. (2019, Nov.) China's new cryptography law: Still no place to hide. China Law Blog. [Online]. Available: https://www.chinalawblog.com/2019/11/chinas-new-cryptography-law-still-no-place-to-hide.html

[6] L. B. Martinkauppi and Q. He, "Performance evaluation and comparison of standard cryptographic algorithms and Chinese cryptographic algorithms," MSc. thesis, 2019. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:bth-18234

[7] A. Sakthivel and R. Nedunchezhian, "Analyzing the point multiplication operation of elliptic curve cryptosystem over prime field for parallel processing," *The International Arab Journal of Information Technology*, vol. 11, no. 4, Jul. 2014.

[8] Z. Zhao and G. Bai, "Ultra high-speed SM2 ASIC implementation," in *Proceedings of IEEE TrustCom*, Beijing, China, Sep. 2014.

[9] X. Du and S. Li, "The ASIC implementation of SM3 hash algorithm for high throughput," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E99.A, no. 7, pp. 1481–1487, 2016.

[10] B. Preneel, V. Rijmen, and A. Bosselaers. (1998, Dec.) Algorithm alley. [Online]. Available: http://www.drdobbs.com/algorithm-alley/184410756

[11] J. Rott. (2012, Feb.) Intel® Advanced Encryption Standard Instructions (AES-NI). [Online]. Available: https://software.intel.com/en-us/articles/intel-advanced-encryption-standard-instructions-aes-ni

[12] L. Bai, Y. Zhang, and G. Yang, "SM2 cryptographic algorithm based on discrete logarithm problem and prospect," in *Proceedings of IEEE CECNet*, Three Gorges, China, Apr. 2012.

[13] D. Feng, Y. Qin, W. Feng, and J. Shao, "The theory and practice in the evolution of trusted computing," *Chinese Science Bulletin*, vol. 59, no. 32, pp. 4173–4189, August 2014. [Online]. Available: https://doi.org/10.1007/s11434-014-0578-x

[14] Z. Zhao and G. Bai, "Exploring the speed limit of SM2," in *Proceedings of IEEE CCIS*, Shenzhen and Hong Kong, China, Nov. 2014.

[15] F. Mendel, T. Nad, and M. Schläffer, "Finding collisions for round-reduced SM3," in *Topics in Cryptology – CT-RSA 2013*. Springer Berlin Heidelberg, 2013, pp. 174–188. [Online]. Available: https://doi.org/10.1007/978-3-642-36095-4_12

[16] T. Ao, Z. He, J. Rao, K. Dai, and X. Zou, "A compact hardware implementation of SM3 hash function," in *Proceedings of IEEE TrustCom*, Beijing, China, Sep. 2014.

[17] Y. Ma, L. Xia, J. Lin, J. Jing, Z. Liu, and X. Yu, "Hardware performance optimization and evaluation of SM3 hash algorithm on FPGA," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7618 LNCS, pp. 105–118, 2012.

[18] F. Liu, W. Ji, L. Hu, J. Ding, S. Lv, A. Pyshkin, and R.-P. Weinmann, "Analysis of the SMS4 block cipher," in *Information Security and Privacy*. Springer Berlin Heidelberg, pp. 158–170. [Online]. Available: https://doi.org/10.1007/978-3-540-73458-1_13

[19] H. Cheng and Q. Ding, "Overview of the block cipher," in *Proceedings of IEEE IMCCC*, Harbin, China, Dec. 2012.

[20] W. Ji, L. Hu, and H. Ou, "Algebraic attack to SMS4 and the comparison with AES," in *Proceedings of ISDF*, Thessaloniki, Greece, Nov. 2009.

[21] L. Li, F. Yang, Y. Pan, W. Mao, and C. Liu, "An implementation method for SM4-GCM on FPGA," in *Proceedings of IEEE IAEAC*, Chongqing, China, Mar. 2017.

[22] M. Dworkin, "Recommendation for block cipher modes of operation: Methods and techniques," NIST, Special Publication 800-38A, Dec. 2001. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf

[23] "TIME(1) Linux User's Manual," http://man7.org/linux/man-pages/man1/time.1.html, March 2019.

[24] "PERF-STAT(1) perf Manual," http://man7.org/linux/man-pages/man1/perf-stat.1.html, March 2019.

[25] N. Jansma and B. Arrendondo, "Performance Comparison of Elliptic Curve and RSA Digital Signatures," 2004, http://www.nicj.net/files/performance_comparison_of_elliptic_curve_and_rsa_digital_signatures.pdf.

[26] A. I. Ali, "Comparison and evaluation of digital signature schemes employed in NDN network," *International Journal of Embedded Systems and Applications*, vol. 5, no. 2, pp. 15–29, Jun. 2015, https://doi.org/10.5121/ijesa.2015.5202.

[27] OpenSSL. SSL_library_init. [Online]. Available: https://www.openssl.org/docs/man1.0.2/man3/SSL_library_init.html

[28] ——. Changelog. [Online]. Available: https://www.openssl.org/news/changelog.html#x44