Design and Implementation of Certificateless Cryptography for IoT Applications

Neam Fares[1], Bo Wang[1], and Spiridon Bakiras[2]
[1]College of Science and Engineering, Hamad Bin Khalifa University, Doha, Qatar
[2]Infocomm Technology Cluster, Singapore Institute of Technology

***Abstract*—This work introduces a cryptographic module for IoT devices that addresses the security vulnerabilities that come with their widespread adoption. Four core cryptographic modules are implemented, including data confidentiality, message integrity, authentication, and secure communication channels. Specifically, the SHA-256 hashing and AES128-CBC/GCM cipher modules are very efficient, with an execution time of just a few μs. For the key exchange functionality, we opted to leverage Elliptic Curve Cryptography (ECC) and, in particular, the BLS12-381 curve, because it enables the implementation of certificateless public-key cryptography. We demonstrate the performance of the Hash to Curve and pairing operations that are required by both the BLS12-381 digital signature scheme and the session key agreement protocol. The pairing operation consists of two main steps, namely, the Miller loop and the final exponentiation. On a 10 MHz clock frequency (simulated in FPGA), a pairing operation between two elliptic curve points takes around 3.68s to complete. Under the BLS12-381 digital signature scheme, the module for signing messages takes 0.76s, while the module for verifying signatures takes 7.35s. Finally, we identified that the parallel point-scalar multiplication technique was the most efficient, and the module for generating a session key on an IoT node takes around 4.03s. To summarize, this paper highlights the importance of addressing the security risks associated with IoT devices and presents a low-cost implementation of hardware-based cryptography for achieving robust security.**

***Index Terms*—Hardware security, Certificateless cryptography, FPGA**

I. Introduction

The Internet of Things (IoT) is a vast network of interconnected physical objects that gather and transmit data over the internet. The integration of sensors and networking technologies enables these devices to exchange real-time data without the need for human intervention. This innovation has facilitated numerous applications, such as smart homes, autonomous cars, smart healthcare, and smart city infrastructure. According to a survey released by Fortune Business Insights, the IoT industry is projected to grow from USD 308.97 billion in 2020 to USD 1,854.76 billion in 2028, indicating the industry's continued expansion [1].

With the widespread adoption of IoT devices, there is an urgent need to address the security risks associated with these devices. However, software-based security solutions may not always be effective and may consume the limited resources of IoT devices. Hardware-based cryptography has emerged as a promising solution for achieving robust security without compromising performance or energy efficiency. Several academic papers have demonstrated the potential benefits of hardware-based cryptography, including improved speed and reduced power consumption [2], [3]. Furthermore, hardware solutions can provide cryptographic primitives in a silicon chip that cannot be tampered by malicious actors. However, due to typical IoT sensors' energy budget and memory capacity limitations, crypto modules such as AES, 3DES, and RSA cannot be accommodated without hardware optimization [4]. Therefore, developing cost-effective and secure crypto solutions that can be seamlessly integrated into low-power IoT devices is crucial for ensuring their security.

Moreover, in contrast to the conventional public key infrastructure (PKI), which requires the issue of certificates by a trusted third party, identity-based cryptography (IBC) is being utilized in this work by enabling IoT devices to produce public keys based on their identities. As a result, IBC makes key management and key distribution simpler, by eliminating the necessity for certificates that may incur a considerable overhead. In conclusion, building a hardware-based crypto module is a crucial step in improving the security of IoT devices.

This paper develops a hardware-based crypto core for IoT devices. The cryptographic modules are implemented in Verilog and consist of encryption and decryption, hashing, digital signature, and session key agreement protocols. These modules protect data confidentiality, ensure message integrity, provide message authentication, and create a secure communication channel. Implementing these hardware-based modules enables higher security levels with minimal energy and hardware overhead. The presented implementation is low-cost and can be easily integrated into various IoT devices to protect against cyber attacks.

II. Design Optimization of the Proposed Hardware-Based Crypto Engine

The proposed hardware-based crypto engine aims to provide secure and efficient cryptographic operations for IoT devices. To achieve this goal, it is essential to carefully design and optimize the cryptographic algorithms. In this section, we will discuss the design optimizations of the proposed hardware-based crypto engine.

A. Advanced Encryption Standard (AES-128)

In this work, we have implemented two encryption modes, namely, Cipher Block Chaining (CBC) and Galois/Counter Mode (GCM), both using AES-128 encryption. The CBC mode chains plaintext blocks with previous ciphertext blocks, while the GCM mode uses counter mode encryption and authentication via Galois field multiplication to generate a unique authentication tag for each message. Moreover, we have incorporated Additional Authenticated Data (AAD) in the GCM mode. This provides an additional layer of authentication to the encrypted data, guaranteeing the confidentiality and integrity of the transmitted data. The idea is to generate an authentication tag that depends on both the plaintext and AAD, making GCM mode more secure compared to other encryption modes like CBC.

B. Secure Hash Algorithm (SHA-256)

In this work, we have implemented SHA-256, a secure cryptographic hash function that generates a unique 256-bit hash value for any input message, using multiple rounds of bitwise operations [5]. The process includes padding, initialization, message expansion, and compression, in which the input message is padded with additional bits, eight 32-bit hash values are initialized, and the message is compressed through several rounds of logical and arithmetic operations. This ensures the resulting hash value is unique to the input message and provides a high level of security.

C. Elliptic Curve Cryptography (ECC)

ECC is now the default choice for implementing public key cryptographic protocols because it is more efficient and secure compared to protocols over finite fields. The following sections focus on the key aspects of the proposed hardware-based ECC engine. These components play a critical role in ensuring the security and efficiency of cryptographic operations on resource-constrained IoT devices.

1) Curve Adopted: Identity-based cryptography (IBC) employs bilinear maps on prime order groups over elliptic curves to eliminate the need for public key certificate storage and management. BLS and BN are two commonly used families of pairing-friendly curves that provide security against the elliptic curve discrete logarithm problem. BLS12-381 was chosen due to its balance between security and performance, as it offers a 128-bit security level and a field modulus p that does not hinder computational efficiency. Additionally, BN curves are more complex than BLS curves [6]. The maximum number of points on the curve is determined by the prime order r of the curve, which is a 255-bit prime.

2) Coordinate System Adopted: ECC involves different coordinate systems for representing points on the curve, including affine and Jacobian coordinates. Jacobian coordinates, a type of projective coordinates, use a triplet (X,Y,Z) to represent a point, which enables faster point addition and doubling operations compared to affine coordinates. Despite being more complex—as the scaling factor Z must be kept track of and inversion operations are required for converting back to affine coordinates—Jacobian coordinates are widely used in implementing all group operations in ECC due to their computational efficiency [7]. The reason is that they allow for point addition and doubling using only modular multiplication, addition, and subtraction, without inversion [8].

3) Montgomery Reduction: Montgomery reduction is a technique used to perform modular multiplication of two large integers, where the modulus is also a large integer. It involves converting the inputs to a special representation, known as Montgomery form, performing the multiplication using only addition, subtraction, and bit-shift operations, and then converting the result back to its original form using an inverse transformation [9]. Montgomery reduction is used to perform all the modular multiplication operations needed in this design.

4) Scalar Multiplication: Scalar multiplication is a fundamental operation in ECC that involves multiplying a point on the curve by a scalar value. Various methods, such as binary and fixed window methods, exist to perform scalar multiplication. The binary method involves doubling and adding points on the curve based on the binary representation of the scalar [7]. The fixed window method is faster for larger scalar values as it reduces the number of doublings and additions, but it requires precomputing and storing base points which can be expensive in terms of memory usage [10]. An alternative approach is the parallel scalar multiplication method that uses two processors to perform point doubling and addition operations concurrently, as described in [11]. After investigating all methods, the improved parallel scalar multiplication method was found to be the fastest.

5) Pairing: The optimal ate pairing is a pairing method for the BLS12-381 curve that is both efficient and secure. It achieves its efficiency by utilizing a twisted Frobenius map to reduce the exponentiation cost. The Miller loop is a key component of the optimal ate pairing, and it involves performing a sequence of mathematical operations on points on the curve, including point additions, point doubling, and a line evaluation. After iterating through the Miller loop, the final exponentiation is performed on the output f to obtain the final result. The output of the Miller loop operation in the optimal ate pairing for the BLS12-381 elliptic curve is an element in the extension field $\mathbb{F}_{p^{12}}$, which can be represented as a pair of 12 coefficients, each being an element of the base field $\mathbb{F}_p$ [12], [13].

D. Digital Signature

The BLS digital signature scheme is an efficient and secure algorithm based on ECC, which generates a signature by multiplying the hash of the message with a secret key. The scheme relies on the hardness of the computational Diffie-Hellman problem to ensure resistance to forgery attacks. The signature is verified using a bilinear pairing operation, with the signature and public key defined in different groups $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. The use of $\mathbb{G}_1$ as the group for the signature leads to smaller signature sizes, faster signing and verification times, and lower storage requirements, making it a desirable choice [14].

1) Hash to Curve: Hash to curve is a function used in the BLS digital signature algorithm to convert messages into points on an elliptic curve, typically $\mathbb{G}_1$, ensuring secure and efficient digital signatures. The MapToGroup method, also known as hash-and-check, involves computing the hash of the message, checking for a point on the curve with the same x coordinate as the hash value, and performing cofactor clearing to ensure the resulting point lies in the main subgroup of the curve. An iterative increment of the hash value may be necessary to find a valid point [15].

2) Signing: The BLS signature scheme is a method that enables a signer to sign a message and generate a signature that can be verified by anyone with access to the public key of the signer. The process of signing a message involves computing the hash of the message using a cryptographic hash function and mapping it to a point on the curve $\mathbb{G}_1$. The resulting point, called $H(M)$, is then multiplied by the private key of the signer, resulting in the signature $\sigma = sk \cdot H(M)$, which is also a point on the curve $\mathbb{G}_1$.

3) Verifying: The BLS signature verification algorithm is used to check the validity of a signature generated using the BLS signature scheme. It takes as input the generator g_2 of $\mathbb{G}_2$, the message M, the signature σ, and the public key pk of the signer, and outputs "1" if the signature is valid and "0" otherwise. The algorithm first computes the hash value of the message M using a cryptographic hash function, which is then used to calculate the pairing of pk and $H(M)$, and the pairing of g_2 and σ. If these two pairings are equal, then the signature is valid, and the algorithm outputs 1; otherwise, it outputs 0.

E. Session Key Agreement

Session key agreement is a process used to establish a shared secret key between two parties to ensure secure communication during a session. The Diffie-Hellman key exchange protocol is a common method used for session key agreement. In IoT, it is necessary to establish a session key between IoT devices and the server to ensure the secure transmission of device measurements to prevent unauthorized access or tampering [16].

Using the BLS12-381 curve and its pairing property, session keys can be established between IoT devices and the server. The server will first generate a secret master key $s < r$. Then, for each IoT device i, the server will assign a secret key $D_i = sQ_i$ in group $\mathbb{G}_1$, where $Q_i = H(ID_i)$, H is a function that maps an ID_i to a point in $\mathbb{G}_1$ (as mentioned in Section II-D1), and ID_i is the unique identifier of the device. This secret key can be hardcoded in each IoT device. A session key can then be established by the following steps:

1) Node i will first choose random $x_i < r$, and send x_iQ_i to the server. The server will also choose $x_s < r$ and send x_sQ_s, where $Q_s = H(ID_s)$. These messages can be exchanged in plaintext.
2) Node i will compute the (symmetric) session key $k = e(x_iD_i, x_sQ_s)$, and the server will compute the same key $k = e(x_iQ_i, x_sD_s)$, where $D_s = sQ_s$ is the server's private key.

In both parties, the computed key is $k = e(Q_i, Q_s)^{sx_ix_s}$, due to the property of the bilinear pairing. This works because the server's elliptic curve points are all in $\mathbb{G}_2$ and the nodes' elliptic curve points are all in $\mathbb{G}_1$. This was chosen after considering the low computational power of IoT devices and that operations of $\mathbb{G}_1$ are much faster. Finally, in practice, the session key will be derived by applying a hash function (such as SHA-256) on the key k. This hash function is determined by the desired size of k. Fig. 1 summarizes how the session key agreement protocol works.

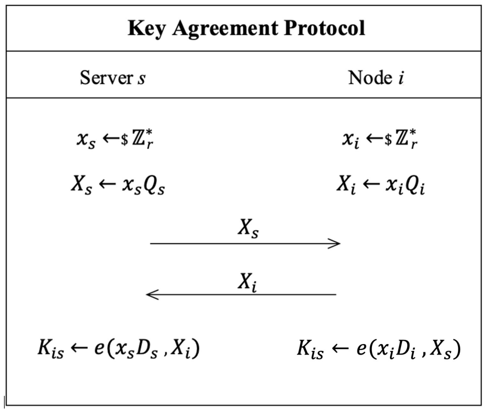
Key Agreement Protocol
Server s
Node i
$x_s \leftarrow\$ \mathbb{Z}_r^*$
$x_i \leftarrow\$ \mathbb{Z}_r^*$
$X_s \leftarrow x_sQ_s$
$X_i \leftarrow x_iQ_i$
X_s
X_i
$K_{is} \leftarrow e(x_sD_s, X_i)$
$K_{is} \leftarrow e(x_iD_i, X_s)$

Fig. 1. Session Key Agreement Protocol

III. FPGA Implementation Results

This section highlights the testing and analysis process of Verilog implemented modules using Icarus and GtkWave. The behavioral simulations were conducted on a personal laptop with a 10MHz clock frequency, and the resulting waveforms were analyzed to determine the time and number of clock cycles required for each operation, providing insight into module performance and efficiency.

The efficiency of scalar multiplication and pairing has a significant impact on the overall performance of ECC. Table I shows the performance of the distinct methods for scalar multiplication, clearly stating that the parallel method performs best for both curves. On the other hand, throughout all the modules, Jacobin coordinates are used to avoid modular inversion operations. However, moving back to affine coordinates requires two inversion operations. Table II shows the time taken and the number of clock cycles required to perform different operations needed for the implemented protocols. Moreover, Fig. 2 shows the waveform of the pairing operation.

TABLE I
Scalar Multiplication Methods Analysis

Method	Time	Clock Cycles
$\mathbb{G}_1$		
Binary Method	0.77s	7721112
Fixed Window Method	0.52s	5204351
Parallel Method	0.48s	4842298
$\mathbb{G}_2$		
Binary Method	3.06s	30633615
Fixed Window Method	2.06s	20633658
Parallel Method	1.92s	19232301

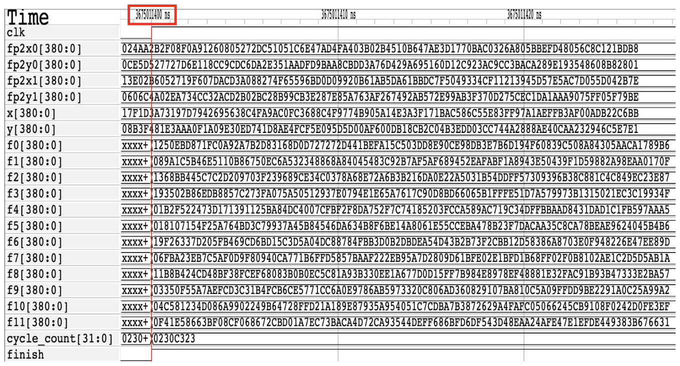

Fig. 2. Pairing Waveform

TABLE II
OPERATION ANALYSIS

Operation	Time	Clock Cycles
Jacobian to Affine in $\mathbb{G}_1$	0.43ms	4311
Jacobian to Affine in $\mathbb{G}_2$	2.02ms	20232
Miller Loop	1.14s	11360796
Final Exponentiation	2.54s	25389316
Pairing	3.68s	36750115
Hash to Curve	0.14s	1352342

Table III summarizes the time and the number of clock cycles taken by all the implemented protocols.

Table IV shows the difference in techniques and parameters used to implement the BLS12-381 curve. All pairing execution times mentioned in Table IV are scaled to 90MHz. Reference [17] proposes an optimized version of the Montgomery reduction algorithm called coarsely integrated operand scanning (CIOS) Montgomery reduction, which employs a coarser scanning process to speed up the conversion of input operands to the Montgomery representation. This results in a reduction in the number of shifts and additions required during the multiplication phase, leading to faster computation times. In contrast, [18] presents the use of a 9-stage pipelined modular multiplier, which utilizes a pipelined architecture to enable multiple modular multiplication operations to be performed in parallel, thereby improving algorithm efficiency. The complexity of these techniques varies, with Montgomery reduction having moderate complexity, CIOS Montgomery reduction having higher complexity due to increased precomputation, and the 9-stage pipelined modular multiplier having the highest complexity due to the hardware resources required for implementing the pipelined architecture. Therefore, our implementation is the least complex.

IV. CONCLUSION

This paper investigates the role of hardware-based cryptography in addressing cybersecurity challenges and vulnerabilities posed by IoT devices. We presented an endpoint-secured chip that achieves hardware-based security and can be embedded in IoT devices. The core cryptographic modules implemented in Verilog include SHA-256, AES-128, BLS12-381 digital signatures, and BLS12-381 session key agreement. They were all analyzed for their efficiency and suitability for real-world applications. This work emphasizes the importance of cryptography in securing IoT devices and presents valuable insights for implementing it efficiently and effectively.

TABLE III
PROTOCOLS ANALYSIS

Protocol	Time	Clock Cycles
SHA-256	6.50µs	66
AES128-CBC Encryption	2.10µs	22
AES128-CBC Decryption	2.90µs	30
AES128-GCM Encryption	6.00µs	60
AES128-GCM Tag Generation	1.70µs	17
BLS12-381 Signing	0.76s	7558411
BLS12-381 Verifying	7.35s	73493726
BLS12-381 Node's Session Key Generation	4.30s	43060920

TABLE IV
COMPARISON WITH EXISTING WORK

Parameter	Banerjee et al. [17]	Masada et al. [18]	This paper
Curve	BLS12-381	BLS12-381	BLS12-381
Scalar Multiplication	Binary Method		Parallel Method
Modular Multiplication	CIOS Montgomery Reduction	9-Stage Pipelined Modular Multiplier	Montgomery Reduction
Clock Frequency	90 MHz	138 MHz	10 MHz
Pairing Execution Time	0.038s	0.000121s	0.408s

ACKNOWLEDGMENT

This publication was made possible by NPRP grant NPRP13S-0122-200135 and the graduate sponsorship research award GSRA8-L-2-0430-21021 from the Qatar National Research Fund (a member of Qatar Foundation).

REFERENCES

[1] Research and Markets, "Global internet of things (iot) market: Trends analysis and forecasts up to 2025," https://www.researchandmarkets.com/reports/5243477/global-internet-of-things-iot-market-trends, 2020, [Accessed on: March 8, 2023].

[2] D. Kim, K. Park, H. Cho, and K. Rhee, "Hardware-based elliptic curve cryptography on arm cortex-m3," *Journal of Cryptographic Engineering*, vol. 9, no. 2, pp. 83–91, 2019.

[3] M. U. Farooq, S. H. I. Jaffery, and A. Maqsood, "Hardware acceleration of advanced encryption standard for iot devices," in *Proceedings of the 2017 International Conference on Open Source Systems and Technologies*. ACM, 2017, pp. 1–6.

[4] K. Mandal, C. Parakash, and A. Tiwari, "Performance evaluation of cryptographic algorithms: Des and aes," in *2012 IEEE Conference on Electrical, Electronics and Computer Science*. IEEE, March 2012, pp. 1–5.

[5] A. H. Gad, S. E. E. Abdalazeem, O. A. Abdelmegid, and H. Mostafa, "Low power and area sha-256 hardware accelerator on virtex-7 fpga," in *2020 2nd Novel Intelligent and Leading Emerging Sciences Conference (NILES)*. IEEE, 2020, pp. 181–185.

[6] S. Scott, M. B. Henry, and S. Josefsson, "Pairing-Friendly Curves," Internet-Draft, March 2008, https://www.ietf.org/archive/id/draft-irtf-cfrg-pairing-friendly-curves-08.html.

[7] I. Blake, G. Seroussi, and N. Smart, *Elliptic Curves in Cryptography*. Cambridge University Press, 2002.

[8] D. Hankerson, J. Lopez Hernandez, and A. Menezes, "Software implementation of elliptic curve cryptography over binary fields," in *Cryptographic Hardware and Embedded Systems-CHES 2000*. Springer, 2000, pp. 1–24.

[9] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of computation*, vol. 44, no. 170, pp. 519–521, 1985.

[10] H. Seo, H. Kim, T. Park, Y. Lee, Z. Liu, and H. Kim, "Fixed-base comb with window-non-adjacent form (naf) method for scalar multiplication," *Sensors*, vol. 13, no. 7, pp. 9483–9512, 2013.

[11] B. Ansari, "Efficient implementation of elliptic curve cryptography." 2005.

[12] F. Vercauteren, "Optimal pairings," *IEEE transactions on information theory*, vol. 56, no. 1, pp. 455–461, 2009.

[13] M. Scott, "Pairing implementation revisited," *Cryptology ePrint Archive*, 2019.

[14] D. Boneh, M. Drijvers, and G. Neven, "Bls multi-signatures with public-key aggregation," *URL: https://crypto. stanford. edu/˜ dabo/pubs/papers/BLSmultisig. html*, 2018.

[15] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *Advances in Cryptology—ASIACRYPT 2001: 7th International Conference on the Theory and Application of Cryptology and Information Security Gold Coast, Australia, December 9–13, 2001 Proceedings 7*. Springer, 2001, pp. 514–532.

[16] M. M. Rathore, E. Bentafat, and S. Bakiras, "Smart home security: a distributed identity-based security protocol for authentication and key exchange," in *2019 28th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2019, pp. 1–9.

[17] U. Banerjee and A. P. Chandrakasan, "A low-power bls12-381 pairing cryptoprocessor for internet-of-things security applications," *IEEE Solid-State Circuits Letters*, vol. 4, pp. 190–193, 2021.

[18] K. Masada, R. Nakayama, and M. Ikeda, "Hardware acceleration of aggregate signature generation and authentication by bls signature over bls12-381 curve," in *2022 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*. IEEE, 2022, pp. 1–3.