PAPER • **OPEN ACCESS**

Math behind smart contracts

To cite this article: Hala S. Omar *et al* 2024 *J. Phys.: Conf. Ser.* **2847** 012002

Journal of Physics: Conference Series **2847** (2024) 012002 doi:10.1088/1742-6596/2847/1/012002

Math behind smart contracts

Hala S.Omar[1*], Tamer O.Diab[2] ,Wageda I.El sobky[3] and M. A. Elsisy[4]

[1] Basic Sciences Department, Benha Faculty of Engineering, Benha University, Benha, Egypt

[2] Electrical Engineering Department, Benha Faculty of Engineering, Benha University, Benha, Egypt

[3] Basic Sciences Department, Benha Faculty of Engineering, Benha University, Benha, Egypt

[4] Basic Sciences Department, Benha Faculty of Engineering, Benha University, Benha, Egypt

[*]E-mail: hala.saeed@bhit.bu.edu.eg

Abstract. This paper presents how smart contracts are based on mathematics. Smart contracts rely on mathematics to guarantee their immutability, security, and enforceability. Cryptographic procedures that are used to safeguard and confirm the contract's implementation, including hash functions and digital signatures, might be used to illustrate this. Mathematical approaches known as hash functions embrace an input of arbitrary size and generate a fixed-size digest or hash. It is impossible to go backwards the process and ascertain the input from the outcome since the outcome is specific to the input. Digital signature techniques are used for digitally signing smart contracts. The most well-known digital signature schemes—Schnorr, Elgamal, and Elliptic curve schemes—that are employed in smart contracts are described in this research.

1. Introduction

In simple terms, smart contracts are blockchain-based algorithms that operate when specific criteria are satisfied [1]. Generally speaking, they are employed in order to streamline the implementation of an agreement to guarantee that both parties can be confident of the result right away, without the need for a middleman or delay [2]. They can be additionally incorporated into different payment methods and digital trades, which may involve bitcoin and other cryptocurrencies like Ethereum and Bitcoin, and they may execute a procedure that starts the next activity when specific criteria are fulfilled [3]. Since the information contained in the blocks of a smart contract is encrypted and saved on a common ledger, the risk of data loss is practically unattainable [4]. Ethereum is the most commonly employed cryptocurrency platform and the most widely adopted smart contract platform. The Solidity programming language was created by the Ethereum community and is intended to be used to create smart contract programs that operate on the Ethereum Virtual Machine (EVM) implementation framework [5]. Nick Szabo first put forth the idea of smart contracts in 1994. Szabo is a cryptographer and law professor who is credited with creating digital currency [6]. Since there was no distributed ledger system or digital infrastructure to enable them back then, there was not much interest in or development around smart contracts. As an aspect of their studies on the Ricardo payment mechanism for asset transfers, Ian Grigg and Gary Howland released a notion known as Ricardian Contracts in 1996, which is also where today's smart contracts get their start [7]. A

decentralized ledger system on a blockchain network was employed for the development of bitcoin cryptocurrency in 2008 [8]. The creation of smart contract software, which adds the conditions of the contract to the blockchain, was made possible by this form of technology. Straightforward "if/when...then..." phrases that are encoded into software on a blockchain are how smart contracts operate [9]. After certain criteria are satisfied and confirmed, a computer network puts the plans into action. These could include issuing tickets, contacting individuals, registering a car, and paying money to the appropriate parties [10]. After the transaction is finished, the blockchain is refreshed. This implies that the deal is completed and that the outcomes are only visible to those who have been given approval [11].

In order to ensure the security, immutability, and enforceability of these deals, smart contracts' mathematics is essential [12]. Hash functions and Digital signatures are two examples of cryptographic procedures that are used to safeguard and confirm the fulfilment of the contract. A contract's conditions are encoded using boolean algebra and logical gates so that a computer can comprehend and process them. Basic elements of digital circuits that carry out Boolean logic operations like AND, OR, and NOT are known as logical gates [13]. By combining these procedures, more intricate logic circuits that represent the terms and circumstances of a contract can be made. The development and evaluation of smart contracts may also make use of additional mathematical strategies and concepts, such as mathematical proof and finite state machines [14]. By using these mathematical techniques, it is possible to make sure that the conditions of the deal are correctly implemented and that the contract will function as planned [15].

In this research, hash functions and a special version (SHA-256) are outlined in section two. In section three, digital signatures and the most famous schemes are explained.

2. Hash Functions

Mathematical approaches known as hash functions accept an input of arbitrary size and provide a fixed-size digest or hash. It is impossible to reverse the procedure and ascertain the input from the outcome because the outcome is distinct from the input [16]. Since every modification to the data leads to in a new hash, hash functions are employed to safeguard and validate data stored on blockchains. To guarantee blockchain security, cryptographers created Secure Hash Algorithms [17].

Secure Hash Algorithms (SHA) are a group of cryptographic operations that are intended to maintain data security. It operates by applying a hash function, which is a method made up compression procedures, bitwise processes, and modular additions, to modify the data [18].

The result of the hash function is a fixed-size string that is completely unlike the initial string. Since these techniques are a single direction functions, it is nearly hard to change them to return the original data once they have been converted into their corresponding hash values [19]. SHA-1, SHA-2, and SHA-3 are a few noteworthy algorithms that were developed with progressively stronger encryption in response to hacker attempts. For example, SHA-0 is no longer relevant because of the extensively known weaknesses [20].

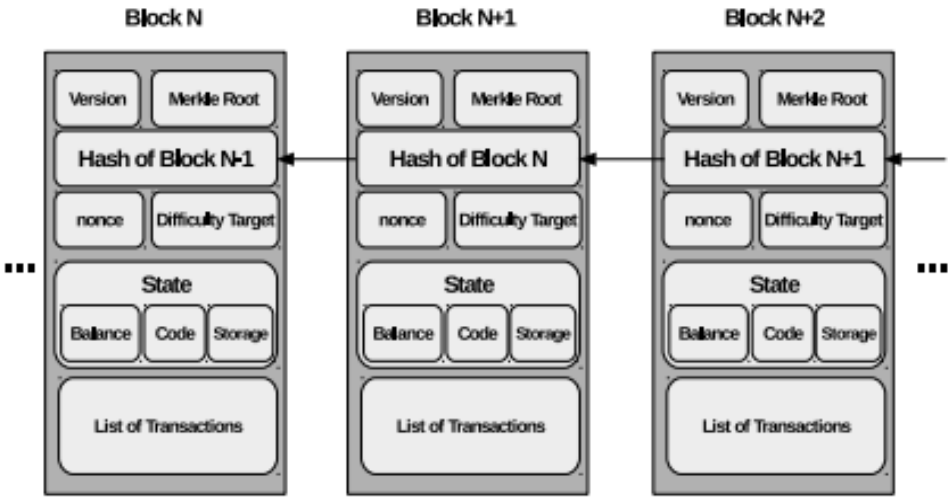

Figure 1. Block chain structure including hash functions.

SHA comes in three versions: SHA-1, SHA-2, and SHA-3. Cryptographers revised the process to create SHA-2 since SHA-1's flaws were revealed. SHA-2 creates two unique hash functions, SHA-256 and SHA-512, using 32- and 64-bit pieces [21]. The remaining half of the algorithm can also be used with these hash functions (SHA-224, SHA-384, SHA-512/224, and SHA-512/256); they are all condensed variants using various constants but follow the same general procedure [22]. The different SHA-2 variants' unit sizes are as follows:

- SHA-2 with output size of 224 /512 bits
- SHA-2 with output size of 256 / 512 bits
- SHA-2 with output size of 384 / 1024 bits
- SHA-2 with output size of 512 / 1024 bits
- SHA-2 with output size of 512 or 224 /1024 bits
- SHA-2 with output size of 512 or256 / 1024 bits

For SHA-512/256, SHA-512/224 and SHA-384, SHA-512, and the filling strategy is practically identical; the only differences are that every single block needs to include data bits of 1024bits and that the last block differs in the ways listed below:[21]

- The second phase inserts zeros until 896 bits are created, rather of using 448 bits.
- Providing 128 bits of the block to be included in the message size is the final step.

In order to include each element of the data in addition to no less than a single digit of padding, the information is divided into multiple blocks as needed. At the end of the last block, the length of 64-bit message is joined.

2.1 SHA-256

SHA-256 works in this manner: [23]

- *Phase One: Preparation of the message by including padding to its length*
  Padding is included to coincide with the length of the original message with 448 modulo 512. As many 0s as needed are inserted after one 1 for padding. Following padding, space is utilized to add the length of the primary text in order to achieve the required length.

- *Phase Two: Initializing hash buffer.*
  The intermediate and final compression function outcomes are stored by SHA-256 in eight 32-bit buffers, designated as a, b, c, d, e, f, g, and h. These encoded (hexadecimal) constants correspond to the primary 32 bits of the fractional portions of the square roots of the initial eight prime integers: 2, 3, 5, 7, 11, 13, 17, and 19.

$$h_0 = 0x6a09e667$$
$$h_1 = 0xbb67ae85$$
$$h_2 = 0x3c6ef372$$
$$h_3 = 0xa54ff53a$$
$$h_4 = 0x510e527f$$
$$h_5 = 0x9b05688c$$
$$h_6 = 0x1f83d9ab$$
$$h_7 = 0x5be0cd19$$

- *Phase Three: Initializing round constants (k)*
  A number of constants are created in the same manner as in phase two. Each value (0–63) is made up of the initial 32 bits of the fractional portions of the cube roots of the first 64 prime integers (2–311).

- *Phase Four: Chunk Loop*
  Every 512-bit "chunk" of data will subsequently be transmitted through the input as we proceed.
- *Phase Five: Compression function*
  The SHA-256 compression technique consists of 64 rounds. The hash buffers (a, b, c, d, e, f, g, and h) are the input of each round. These buffers are computed to supply hash buffers with fresh values for the input's next round. As seen in Figure 1, the value w_i of 32-bit is supplied for each round. It is obtained from a block of 512-bit using a message schedule.
  Moreover, an extra constant K_i is required for each round. Its constant value, which ranges from 1 to 64, is determined by the round number. To construct H_i, the result of the most recent round is added to the hash input of the previous round.
- *Phase Six: Output Digest*
  The 256-bit message digest is the result of the last 512-bit block.
  In Figure 2, the SHA-256 algorithm is demonstrated.

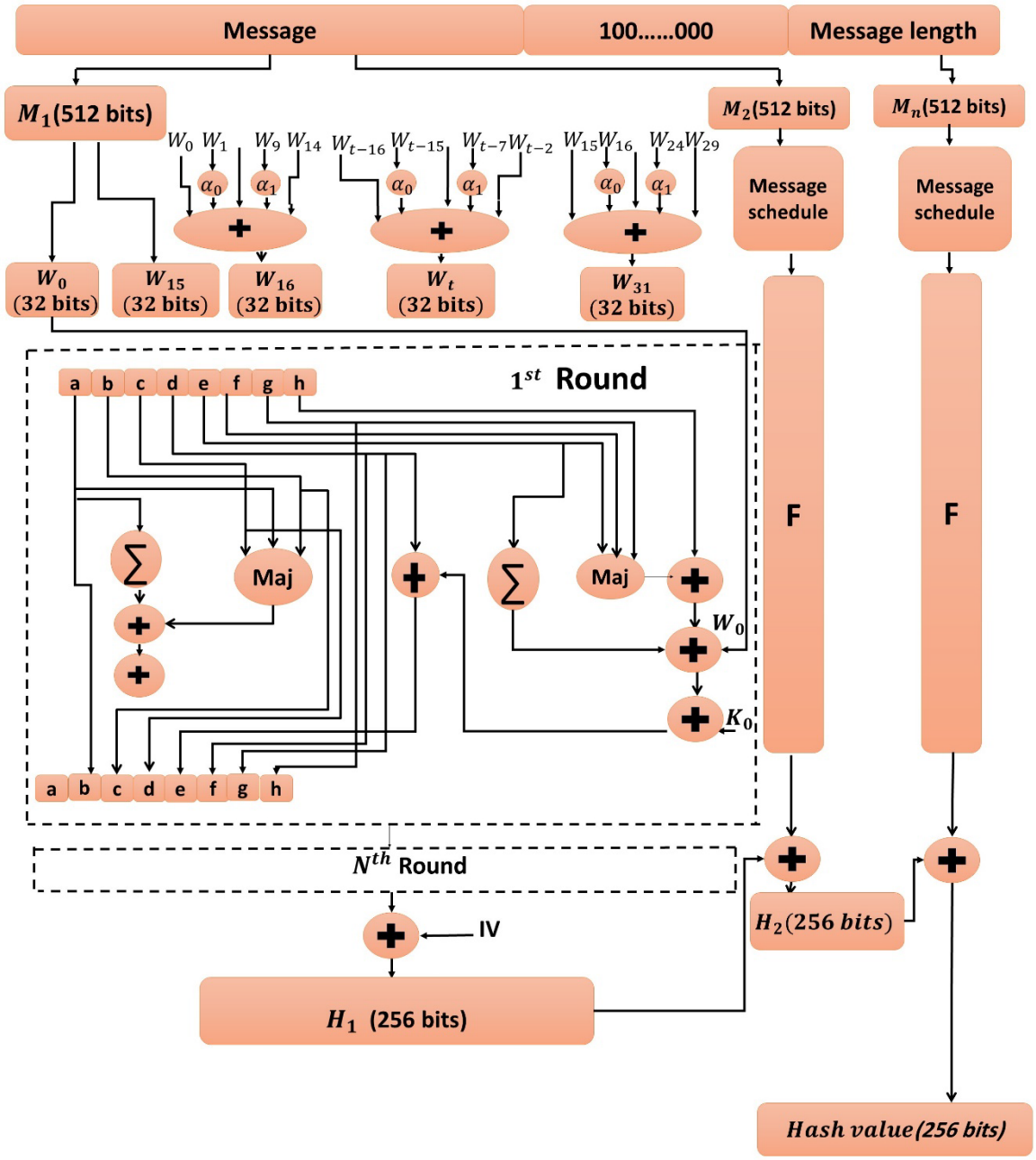

Figure 2. SHA-256 algorithm

3. Digital Signatures

A mathematical strategy called a digital signature can be applied to guarantee the reliability and authenticity of a digital document, software, or transmission [24]. Although it is a digital equivalent of a printed document or embossed sign, it provides a lot more intrinsic security. It also attempts to tackle the problem of impersonation and hacking in online conversations. It can offer proof of the author, status, and ownership of emails, documents or transactions [25]. They can also be used by signers to verify that they gave their informed permission. In

many countries, including the US, they have the same legal force and effect as conventional handwritten printed signatures. Signing technology, like an email software, provides a single direction hash of the online data to be verified in order to create a digital signature [26]. It is impossible for the signer to say they never signed anything cause the document and the signer are uniquely identified and linked by the digital signature. We call this attribute non-repudiation. Digital signatures are widely used to sign smart contracts [27].

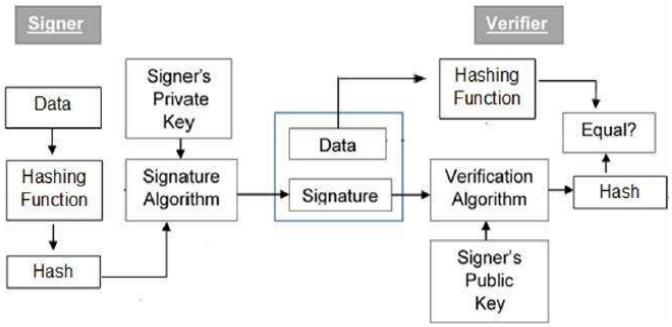

Figure 3. A model of digital signature.

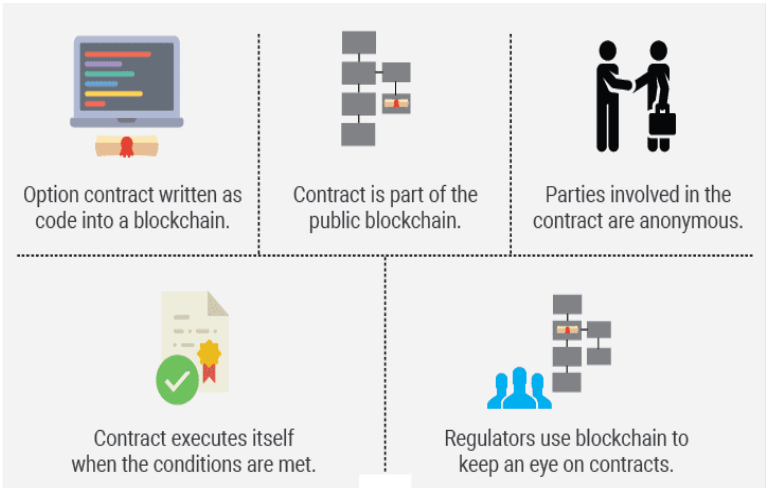

Figure 4. Smart contract process.

Public key cryptography, sometimes referred to as asymmetric cryptography, is the backbone of digital signatures. A methodology of public key, such as Rivest-Shamir-Adleman (RSA), is used to preserve two keys, one secret and the other public, resulting in a pair of mathematically related keys. Several digital signature algorithms have been developed based on this concept. However, because of their flaws, only a few of these algorithms were appropriate for application in smart contracts, which included lengthy processing times and insufficient security [20].

The following procedures are the most widely used and appropriate ones for digital signatures used in smart contracts:

3.1 Schnorr digital signature algorithm

Claus Schnorr has provided a description of this algorithm [28]. It is well known for being straightforward and is among the first digital signature techniques with security established on

the obstinacy of particular discrete logarithm issues [29]. It works well and generates signatures that are succinct. In Figure 5, the algorithm is displayed.

A) Choosing parameters

- Within the prime order q group G with generator g, all participants in the signature method concur, the discrete log issue is assumed to be hard. Usually, a group of Schnorr is implemented.
- All individuals accept the encoded hash function $H: \{0,1\}^* \rightarrow \mathbb{Z}_q$.

B) Key generation

- u, the signing key, which is secret, is picked from $\mathbb{Z}_q$.
- $t = g^u mod q$ is the public key for verification.

C) Signing

For the purpose of making a sign, M:

- A random selection is made from the allowed range for l.
- Suppose

$$\boldsymbol{w = g^l} \tag{1}$$

- Consider

$$\boldsymbol{z = H(w||M)} \tag{2}$$

wherein the bit string representation of the concatenation symbol, ||, is illustrated.

- Consider

$$\boldsymbol{s = l - uz} \tag{3}$$

- The pair of the signature is (s, z).
- Consider that $s, z \in \mathbb{Z}_q$; if $q < 2^{160}$, then 40 bytes can hold the signature representation.

D) Verifying

- Suppose

$$\boldsymbol{w_v = g^s t^z} \tag{4}$$

- Suppose

$$\boldsymbol{z_v = H(w_v||M)} \tag{5}$$

- The signature is authenticated if $z_v = z$.

E) Correctness Proof

- It is quite easy to demonstrate that the signed and confirmed messages are the same if $z_v = z$

$$\boldsymbol{w_v = g^s t^z = g^{l-uz} g^{uz} = g^l = w} \tag{6}$$

and therefore $$\boldsymbol{z_v = H(w_v||M) = H(w||M) = z} \quad (7)$$

- G, g, q, t, s, z, w are the public elements.
- l, u are the private elements.
- All this indicates that a properly signed message will pass verification; a safe signing mechanism needs to include a lot more features.

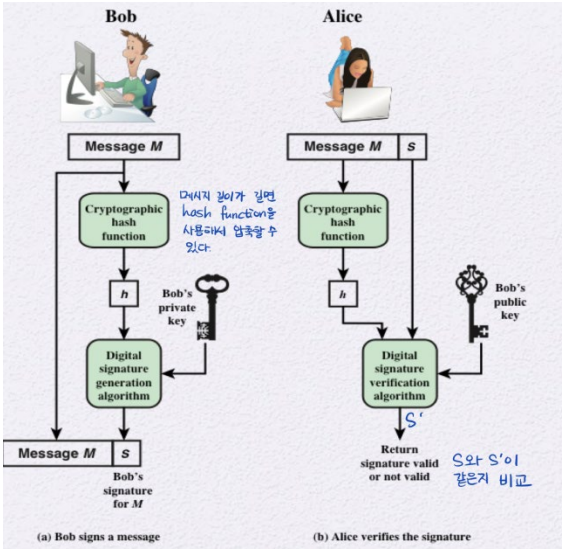

Figure 5. Schnorr signature algorithm.

3.2 Elgamal Signature Algorithm

El Gamal signature algorithm has been suggested depending on the discrete logarithm computing complexity [30]. It was first designed in 1985 by Taher Elgamal. Figure 5 [31] displays the algorithm.

A) Key generation

The process of creating keys has two parts. The first stage is to choose algorithm components that other system users can share, and the second is to estimate a single key pair for a particular user.

B) Parameter generation

- A length of key N is stated.
- p is selected as an N -bit prime number.
- The outcome length of an encoded hash function, H, is set to L bits. If L > N, just the leftmost N bits are used in the hash outcome.
- A generator of $g < p$ the multiplicative group of integers modulo p, $\mathbb{Z}^*_p$ is chosen.
- The components of the algorithm are (p, g) .Members of the system may have these components in common.

C) Per-user keys

The next phase uses an assortment of elements to compute the key pair for a particular user:

- A random integer u is selected from $\{1, \ldots\ldots, p-2\}$

- Determine

$$\boldsymbol{t = g^u \mathrm{mod} p} \tag{8}$$

- u is the hidden key and t is the shared key.

D) Signing

The following procedure generates a sign for a message m:

- An unpredictability integer, l, is selected from {2, ..., p-2}, where l is almost prime to p-1.
- Determine

$$\boldsymbol{w = g^l \mathrm{mod} p} \tag{9}$$

- Estimate

$$\boldsymbol{s = (H(m) - uw) l^{-1} \mathrm{mod} (p - 1)} \tag{10}$$

- You have to start a new with a new random l in the rare case if s=0.
- The signature will be (w, s) .

E) Verifying a signature

To ascertain whether a signature for a message m is authentic, take the subsequent actions:

- Ensure that $0 < w < p$ and $0 < s < p - 1$.
- The authenticity of the signature is contingent upon
-

$$\boldsymbol{g^{H(m)} \equiv t^w w^s \mathrm{mod} p} \tag{11}$$

F) Correctness

- The algorithm is accurate since a signature made with the signing mechanism will always be recognized by the verifier.
- The estimation of s during the generation of a signature conveys.

$$\boldsymbol{H(m) \equiv uw + sl \mathrm{mod} (p - 1)} \tag{12}$$

Taking into account that g is significantly prime to ,

$$g^{H(m)} \equiv g^{uw+sl} (\mathrm{mod} p) \tag{13}$$

$$\equiv (g^u)^w (g^l)^s (\mathrm{mod} p) \tag{14}$$

3.3 Elliptic Curve Signature Algorithm.

Elliptic-curve cryptography is implemented by the Elliptic Curve Digital Signature algorithm (ECDSA), which provides a replacement to the Digital Signature algorithm (DSA) [32].

It's crucial to comprehend the fundamentals of the Elliptic Curve in order to comprehend Elliptic Curve signatures more fully [33]. A planar algebraic curve that is stated by the following equation is called an elliptic curve.

$$y^2 = x^3 + ax + b \tag{15}$$

An elliptic curve generally has the appearance depicted in Figure 6. When a straight line is drawn to cross an elliptic curve, it can intersect nearly three spots. It is evident that the elliptic curve is identical about the x-axis. This feature is necessary for the technique.

$$\equiv (t)^w (w)^s (\mathrm{mod} p) \tag{16}$$

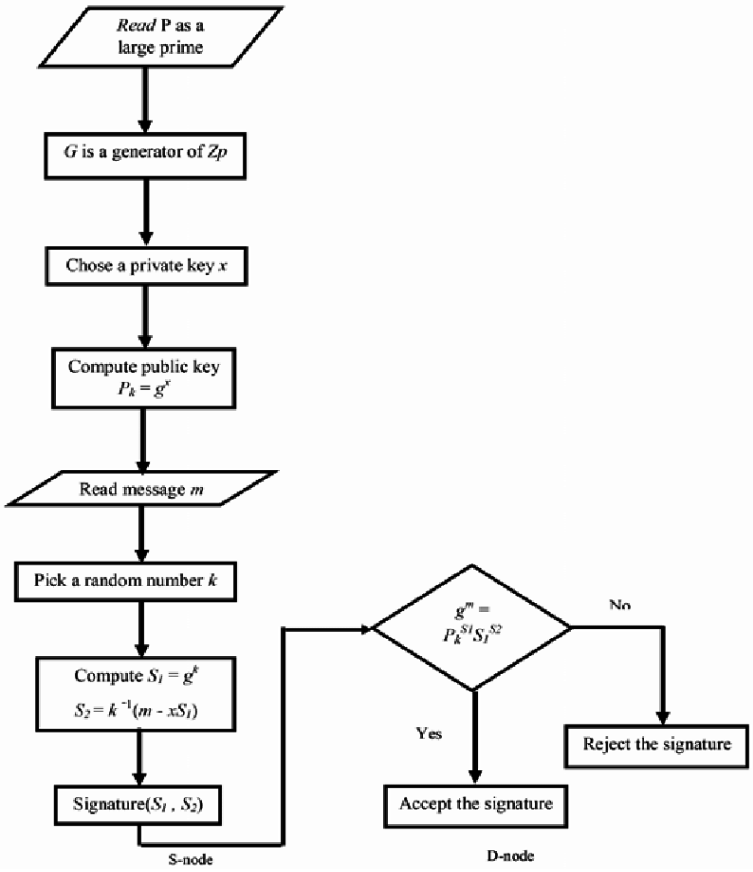

Figure 6. Elgamal signature algorithm.

For the purpose of simplicity and practical procedure implementation, we will only consider four parameters: two private values, a and b; one prime, P; and G (a primitive root of P). P and G are integers that are known to the public. People (let's say Alice and Bob) choose hidden values a and b, create a key, and transfer it in public (let's say $x = G^a \ mod \ p$) and (let's say $y = G^b \ mod \ p$), respectively. People are aware of the integers P and G.

After receiving the key and using it to build a secret key, the other person can encrypt ($k_a = y^a \mathrm{mod} P$) and ($k_b = y^b \mathrm{mod} P$) with the exact same secret key.

A) Key generation

- A public key curve point, $QA = dA \times G$, which is the elliptic curve's base point, and a place on the curve that provides a large prime order n portion, $n \times G = O$, where O is the identity component, make up the key combination that Alice creates. A random selection is made for the hidden key integer dA within the interval $[1, n - 1]$. A scalar multiplier of $\times$ is used for multiplying points on an elliptic curve.

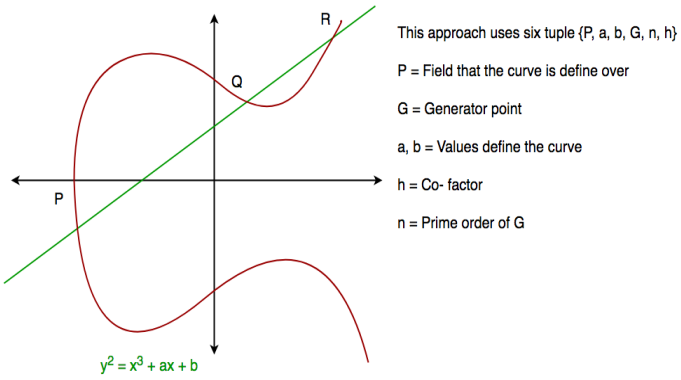

Figure 7. Elliptic curve points.

B) Signing

- Here $z = Hash(m)$, m is the message that needs to be signed, and HASH is an encoded hash function—similar to SHA-2—with an outcome that is converted to an integer.
- Let E represent the $L_n$leftmost bits of z, where L_n denotes the group order n 's bit length. Remember that E cannot be longer than n, although it can be greater.
- A cryptographically secure random number, l, is selected from $[1, n - 1]$.
- Determine the approximate location of the curve.

$$(\boldsymbol{a_1}, \boldsymbol{b_1}) = \boldsymbol{l} \times \boldsymbol{G} \tag{17}$$

- Estimate

$$w = a_1 \bmod n \tag{18}$$

- Refer back to step 3 if $w = 0$.
-
- Determine

$$s = l^{-1}(E + wd_A) \bmod n \tag{19}$$

- If $s = 0$, return to step 3.
- The signature item is (z, s). Thus, $(w, -s \ mod \ n)$ is a trustworthy signature.

C) Verification

- Confirm that the values of w and s in $[1, n - 1]$ are integers. The signature is null and void otherwise.

- Determine

$$z = Hash(m), \tag{20}$$

- where the signature-generating technique is the same as HASH.
-
- Assume that E be the L_n leftmost bits of z .
- Determine

$$t_1 = Es^{-1} \bmod n \tag{21}$$

- And

$$t_2 = ws^{-1} \bmod n \tag{22}$$

- Determine the approximate location of the curve.
-

$$(a_1, b_1) = t_1 \times G + t_2 \times Q_A \tag{23}$$

- If $(a_1, b_1) = O$ then the signature is invalid.
- If $w \equiv a_1 \bmod n$ The signature is considered credible; if not, it is invalid.

D) Correctness

$$\boldsymbol{t_1 = sz^{-1} \bmod n} \tag{24}$$

$$\boldsymbol{t_2 = -wz^{-1} \bmod n} \tag{25}$$

$$\boldsymbol{Y = t_1 P + t_2 Q} \tag{26}$$

$$\boldsymbol{Y = a \bmod n} \tag{27}$$

$$\boldsymbol{\because Q_A = d_A \times G} \tag{28}$$

$$\boldsymbol{\therefore V = t_1 \times G + t_2 d_A \times G} \tag{29}$$

$$\boldsymbol{V = (t_1 + t_2 d_A) \times G} \tag{30}$$

$$\boldsymbol{V = (Es^{-1} + wd_A s^{-1}) \times G} \tag{31}$$

$$\boldsymbol{V = (E + wd_A) s^{-1} \times G} \tag{32}$$

$$\boldsymbol{V = (E + wd_A)(E + wd_A)^{-1} (l^{-1})^{-1} \times G} \tag{33}$$

$$\boldsymbol{V = l \times G} \tag{34}$$

This is a validation of the last phase in accordance with the *w* idea. In Figure 8, the algorithm is displayed.

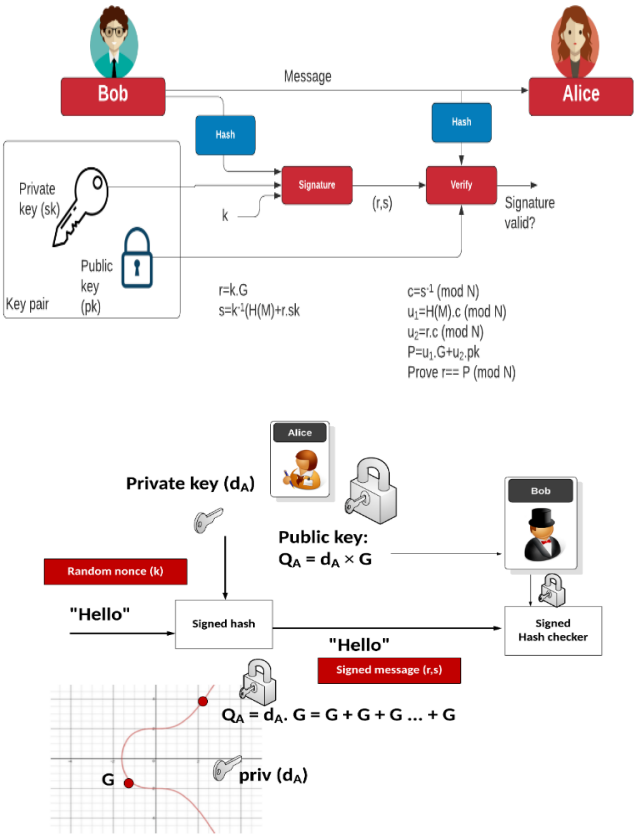

Figure 8. Elliptic curve strategy.

Conclusion

Mathematics plays a great role in smart contracts. Cryptographic technologies like digital signatures and hash functions may be employed to represent it. These techniques are used to ensure security, immutability, and enforceability of these contracts. Hash functions are used to secure the blockchain where smart contracts are stored while digital signatures are used to sign the contract. In this study, SHA-256 is explained as a version of hash algorithms that is mostly used in smart contract. Also, the most familiar digital signature schemes used in these contracts are presented. These schemes include Schnorr scheme, Elgamal scheme and Elliptic curve scheme. Future work may involve some modifications to secure hash algorithms and digital signature schemes to enhance the performance of them and increase their security in smart contracts.

References

[1] Zheng, Z., Xie, S., Dai, H. N., Chen, W., Chen, X., Weng, J., & Imran, M. (2020). An overview on smart contracts: Challenges, advances and platforms. Future Generation Computer Systems, 105, 475-491.
[2] Ante, L. (2021). Smart contracts on the blockchain–A bibliometric analysis and review. Telematics and Informatics, 57, 101519.
[3] Hewa, T. M., Hu, Y., Liyanage, M., Kanhare, S. S., & Ylianttila, M. (2021). Survey on blockchain-based smart contracts: Technical aspects and future research. IEEE Access, 9, 87643-87662.
[4] John, K., Kogan, L., & Saleh, F. (2023). Smart contracts and decentralized finance. Annual Review of Financial Economics, 15, 523-542.
[5] Kirli, D., Couraud, B., Robu, V., Salgado-Bravo, M., Norbu, S., Andoni, M., ... & Kiprakis, A. (2022). Smart contracts in energy systems: A systematic review of fundamental approaches and implementations. Renewable and Sustainable Energy Reviews, 158, 112013.
[6] Metcalfe, W. (2020). Ethereum, smart contracts, DApps. Blockchain and Crypt Currency, 77.
[7] H.Saeed, Elsisi, M. A., Diab, T. O., El Sobky, W. I., Abdel-Wahed, M. S., & Mahmoud, A. K. (2023, July). Famous Digital Signatures Used In Smart Contracts. In 2023 International Telecommunications Conference (ITC-Egypt) (pp. 649-656). IEEE.
[8] Balcerzak, A. P., Nica, E., Rogalska, E., Poliak, M., Klieštik, T., & Sabie, O. M. (2022). Blockchain technology and smart contracts in decentralized governance systems. Administrative Sciences, 12(3), 96.
[9] Fauziah, Z., Latifah, H., Omar, X., Khoirunisa, A., & Millah, S. (2020). Application of blockchain technology in smart contracts: A systematic literature review. Aptisi Transactions on Technopreneurship (ATT), 2(2), 160-166.
[10] Oliva, G. A., Hassan, A. E., & Jiang, Z. M. (2020). An exploratory study of smart contracts in the Ethereum blockchain platform. Empirical Software Engineering, 25, 1864-1904.
[11] Xu, Y., Chong, H. Y., & Chi, M. (2021). A review of smart contracts applications in various industries: a procurement perspective. Advances in Civil Engineering, 2021, 1-25.
[12] Tolmach, P., Li, Y., Lin, S. W., Liu, Y., & Li, Z. (2021). A survey of smart contract formal specification and verification. ACM Computing Surveys (CSUR), 54(7), 1-38
[13] W. Alsobky, H. Saeed, and A. N. Elwakeil. Sep. 2020.''Different types of attacks on block ciphers,'' Int. J. Recent Technol. Eng. (IJRTE), vol. 9, no. 3, pp. 28–31.
[14] Bottoni, P., Gessa, N., Massa, G., Pareschi, R., Selim, H., & Arcuri, E. (2020). Intelligent smart contracts for innovative supply chain management. Frontiers in Blockchain, 3, 52.
[15] Hewa, T., Ylianttila, M., & Liyanage, M. (2021). Survey on blockchain based smart contracts: Applications, opportunities and challenges. Journal of network and computer applications, 177, 102857.
[16] Mittelbach, A., & Fischlin, M. (2021). The theory of hash functions and random oracles. An Approach to Modern Cryptography, Cham: Springer Nature.
[17] Tutueva, A. V., Karimov, A. I., Moysis, L., Volos, C., & Butusov, D. N. (2020). Construction of one-way hash functions with increased key space using adaptive chaotic maps. Chaos, Solitons & Fractals, 141, 110344.
[18] Ali, A. M., & Farhan, A. K. (2020). A novel improvement with an effective expansion to enhance the MD5 hash function for verification of a secure E-document. IEEE Access, 8, 80290-80304.
[19] Al-Layla, H. F., Ibraheem, F. N., & Hasan, H. A. (2022). A Review of Hash Function Types and their Applications. Wasit Journal of Computer and Mathematics Science, 1(3).
[20] Gnatyuk, S., Kinzeryavyy, V., Kyrychenko, K., Yubuzova, K., Aleksander, M., & Odarchenko, R. (2020). Secure hash function constructing for future communication systems and networks. In Advances in Artificial Systems for Medicine and Education II 2 (pp. 561-569). Springer International Publishing.
[21] Paul, L. S. J., Gracias, C., Desai, A., Thanikaiselvan, V., Suba Shanthini, S., & Rengarajan, A. (2022). A novel colour image encryption scheme using dynamic DNA coding, chaotic maps, and SHA-2. Multimedia Tools and Applications, 81(26), 37873-37894.
[22] Zhang, Y., He, Z., Wan, M., Zhan, M., Zhang, M., Peng, K., ... & Gu, H. (2021). A new message expansion structure for full pipeline SHA-2. IEEE Transactions on Circuits and Systems I: Regular Papers, 68(4), 1553-1566.
[23] Wang, J., Liu, G., Chen, Y., & Wang, S. (2021). Construction and analysis of SHA-256 compression function based on chaos S-box. IEEE Access, 9, 61768-61777.

[24] Vora, J., DevMurari, P., Tanwar, S., Tyagi, S., Kumar, N., & Obaidat, M. S. (2018, July). Blind signatures based secured e-healthcare system. In 2018 International conference on computer, information and telecommunication systems (CITS) (pp. 1-5). IEEE.
[25] Pooja, M., & Yadav, M. (2018). Digital signature. International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), 3(6), 71-75.
[26] Fang, W., Chen, W., Zhang, W., Pei, J., Gao, W., & Wang, G. (2020). Digital signature scheme for information non-repudiation in blockchain: a state of the art review. EURASIP Journal on Wireless Communications and Networking, 2020(1), 1-15.
[27] Gatteschi, V., Lamberti, F., Demartini, C., Pranteda, C., & Santamaría, V. (2018). Blockchain and smart contracts for insurance: Is the technology mature enough?. Future internet, 10(2), 20.
[28] Ferreira, A. (2021). Regulating smart contracts: Legal revolution or simply evolution?. Telecommunications Policy, 45(2), 102081.
[29] Fleischhacker, N., Jager, T., & Schröder, D. (2014). On tight security proofs for Schnorr signatures. In Advances in Cryptology–ASIACRYPT 2014: 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, ROC, December 7-11, 2014. Proceedings, Part I 20 (pp. 512-531). Springer Berlin Heidelberg.
[30] Yu, G. (2020). Simple Schnorr signature with Pedersen commitment as key. Cryptology ePrint Archive.
[31] Khadir, O. (2013). New variant of ElGamal signature scheme. arXiv preprint arXiv:1301.3258.
[32] Luo, Y., Ouyang, X., Liu, J., & Cao, L. (2019). An image encryption method based on elliptic curve elgamal encryption and chaotic systems. IEEE Access, 7, 38507-38522.
[33] Mehibel, N., & Hamadouche, M. H. (2020). A new enhancement of elliptic curve digital signature algorithm. Journal of Discrete Mathematical Sciences and Cryptography, 23(3), 743-757.