SEAL: Secure and Efficient Authentication using Linkage for Blockchain Networks

Hsiang-Jen Hong* Sang-Yoon Chang† Wenjun Fan‡ Simeon Wuthier† Xiaobo Zhou†
* Western Washington University, Bellingham, WA, USA, hongh6@wwu.edu
† University of Colorado Colorado Springs, Colorado Springs, CO, USA, {schang2, swuthier, xzhou}@uccs.edu
‡ School of Advanced Technology, Xi'an Jiaotong-Liverpool University, Suzhou, Jiangsu, China, Wenjun.Fan@xjtlu.edu.cn

2024 27th Conference on Innovation in Clouds, Internet and Networks (ICIN) | 979-8-3503-9376-7/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICIN60470.2024.10494470

Abstract—**The cryptocurrency's permissionless and large-scale broadcasting requirements prohibit the traditional authentication implementation on the blockchain's underlying peer-to-peer (P2P) networking. Thus, blockchain networking implementations remain vulnerable to networking integrity threats such as spoofing or hijacking. We design Secure and Efficient Authentication using Linkage (SEAL) to build connection security for permissionless blockchain networking. SEAL uses the linkage between the packets for a symmetric operation, in contrast to the traditional authentication approach relying on identity-credential-based trust. To make it appropriate for cryptocurrency networking, SEAL utilizes the packet header, protects the end-to-end connection, and separates the online process (based on a simple mixing operation) vs. the offline process (the in-advance construction of the pseudo-random number chain) so that the real-time overhead is minimal for greater efficiency and practicality. We implement SEAL on a working Bitcoin node and show that SEAL works and provides minimal overhead. Furthermore, we use a networking simulator to simulate the Bitcoin Mainnet and analyze the SEAL impact on the block propagation delay on the network, e.g., SEAL yields 2.04 times smaller delay and 1.25 times smaller delay in block propagation than HMAC and ChaCha20-Poly1305, respectively.**

Index Terms—**Bitcoin, Blockchain, Cryptocurrency, Permissionless, Distributed networking, Authentication, P2P networking**

I. Introduction

Cryptocurrencies such as Bitcoin use blockchain to simultaneously protect the integrity of transactions and support permissionless and censorless transactions. Blockchain networking based on peer-to-peer (P2P) networking is critically important because the networking provides the information (including the latest blocks and transactions) to enable the rest of the blockchain and cryptocurrency operations.

However, P2P networking remains vulnerable to networking integrity threats such as spoofing or hijacking due to the lack of cryptographic protections for two reasons. First is the permissionless requirement of the cryptocurrency application, which enables *any* user to join the network or create new keys/accounts for transactions, including dynamically creating multiple keys/accounts for anonymity and censorless purposes [1]. The permissionless requirement thus forbids the use of a centralized authority such as the traditional public-key infrastructure (PKI), which could have controlled which users join the network (violating the permissionless) but could have provided the networking authentication for the key (providing the networking root of trust for the traditional authentication approaches). Instead, the users generate and use self-certified keys which lack authority-based certification. Second, the distributed computing and consensus protocol of blockchain requires a large size of networking participants to be resilient against an attacker compromising the participants, e.g., 51% attack. Since a larger network increases the networking overhead for blockchain synchronization and transaction/block broadcasting, the standard authentication techniques to protect the networking source integrity become cost-prohibitive. The current blockchain implementations, e.g., Bitcoin, thus remain vulnerable with no authenticity protections. The networking packets are transmitted in plaintext and without integrity checks, and an unauthorized attacker can masquerade as another peer or hijack an existing connection. The networking threats on cryptocurrency networking build on such threat vectors, including those for routing manipulations for hijacking and partitioning [2] and disconnections [3], [4], [5].

We thus build an authentication scheme, called SEAL (Secure and Efficient Authentication using Linkage), for TCP-based permissionless blockchain networking to defend against spoofing and hijacking against an existing connection. To address the aforementioned blockchain-unique challenges, we build security without relying on the explicit trust (traditionally provided by the centralized PKI) and use the linkage across the packets for the trust-dependency, i.e., the legitimate current packet is linked to the previous packet (while the attacker spoofing the ID credentials cannot provide such linkage-based information). Consider a case where Alice and Bob use our scheme, and Mallory masquerades as Bob, having compromised the identity credentials, e.g., IP address and port number. While Mallory can send packets as Bob, Alice can distinguish the two networking streams and identify those packets from the original Bob vs. somebody different from the original Bob (Mallory in this case). Such reliance on the linkage/dependency across the history of the networking packets distinguishes our work from the standard Message Authentication Code (MAC), as discussed in Section II.

A general overview of SEAL is described as follows: The sender, utilizing SEAL, writes data (generated from the original packets and numbers provided by secure pseudo-random number chains) into the TCP options field for authentication purposes. Our construction of secure pseudo-random number

Authorized licensed use limited to: Università di Bari Aldo Moro. Downloaded on September 17,2025 at 10:46:41 UTC from IEEE Xplore. Restrictions apply.

chains is inspired by Output Feedback (OFB) and is similar in that it enables stream-cipher-like operations. It also facilitates the construction of an in-advance/offline Pseudo-Random Number Generator (PRNG), no longer serving as the bottleneck for real-time authentication. The receiver, employing SEAL, verifies the packet while addressing the challenges of using the correct numbers in the secure pseudo-random number chains, as discussed in Section V-D. SEAL, therefore, can distinguish between a legitimate peer connection and an adversary attempting to inject packets to impersonate that peer.

We prioritize the practicality and appropriateness of blockchain networking and describe the unique challenges in cryptocurrency networking. The lightweight design is one of our design goals because the overhead raised by the authentication can affect the block propagation delay. More specifically, the cascading effect of the additional overheads using authentication schemes could significantly impact the block propagation performance (as shown in Section VI-B). More importantly, SEAL is compatible with the current TCP protocol (i.e., it does not use a separate data field) and is significantly more lightweight than MAC. We distinguish the offline process for the pseudo-random generation before the message communications vs. the online process for mixing/utilizing the pseudo-random output with the packet to achieve superior efficiency.

II. Challenges in Blockchain Networking

There are unique challenges from blockchain networking distinct from traditional networking: the permissionless requirement and the broadcasting networking overhead. These challenges forbid the use of standard networking security techniques and motivate our work. First, cryptocurrency applications require permissionless operations to forgo a centralized authority. This requirement prohibits using a centralized public-key infrastructure (PKI) since it can yield centralization in identity/key management. Forgoing PKI challenges the traditional explicit trust-based security, specifically those utilizing the certificate-based public key as the root of trust. Second, the networking overhead escalates quickly as the number of participants grows. Support for greater network size is inherently required for the resiliency of distributed computing. With smaller sizes, e.g., new coins/chains, the attacker's feasibility to compromise the majority of the network to launch a 51% attack and double spending becomes greater and, thus, greater security risk against the transaction integrity. There has been reported real-world incidents of 51% attack and double spending on newly beginning cryptocurrencies [6], [7] but not on the mature cryptocurrencies such as Bitcoin and Ethereum, which already have healthy sizes of networking and consensus participants. In Bitcoin, there is a conservative estimate of 49,000 available nodes as of July 2023 [8]. For each node, there can be up to 128 connections using the default protocol and possibly even higher with implementation modification, e.g., 708 connections [9]. The networking overhead increases even further because of the broadcasting nature of networking which reaches all the networked nodes so that everybody can get synchronized in the ledger.

The permissionless blockchain and cryptocurrencies encounter significant challenges when it comes to cryptographic protection for authentication in networking. This presents a notable contrast to the prevailing approach of employing hash functions and public-key signatures to ensure transaction integrity at the OSI application layer. Interestingly, even MACs, which demonstrate efficiency in various networking applications, remain unutilized in cryptocurrency networking. The adoption of Transport Layer Security (TLS) mechanisms for blockchain networking is also impeded. TLS encompasses encryption and authentication; the encryption and decryption of entire packets used in blockchain networking can significantly impact broadcasting performance. Consequently, present-day communication in blockchain networking, including broadcasting Bitcoin messages, necessitates transmitting data in plaintext via TCP connections only.

III. Threat Model

Our work focuses on the source integrity of networking. We focus on the threats against an already existing connection, and the connection establishments are P2P and distributed. An attacker succeeds in its attack if the attacker can spoof as a victim and inject messages as the victim. We consider the attacker who can eavesdrop on the connection, process the plaintext connection, keep track of the exchanged packets, construct networking packets as a different peer, and inject messages. We do not assume that the attacker has the capability to perform route manipulation to make itself have a privileged position in the middle of the target node and the Bitcoin Mainnet. In other words, the consideration of the man-in-the-middle attacker is out of our threat model.

Our attack prototype based on a Bitcoin node shows that an unauthorized attacker can spoof the IP address and the port number and counterfeit the packet's header information, e.g., the sequence number field, for such capabilities. As long as the attacker can sniff the plaintext packets, it can learn the expected sequence number and acknowledgment number to craft acceptable TCP packets to inject misbehaving messages. Equipped with such capabilities, an attacker can masquerade as the victim's peer and hijack an existing connection. We also analyze the security of our scheme in computational security and assume that the attacker is computationally bounded. The security of our scheme assumes the cryptographic primitives and assumptions in one-way hash function, i.e., no known algorithm which efficiently breaks the hash function by finding collision inputs.

IV. System Model and Framework

A. System Model

Our goal is to secure pairwise communication on the P2P network and to make it practical for cryptocurrency applications. As described in Section II, the participating nodes

Authorized licensed use limited to: Università di Bari Aldo Moro. Downloaded on September 17,2025 at 10:46:41 UTC from IEEE Xplore. Restrictions apply.

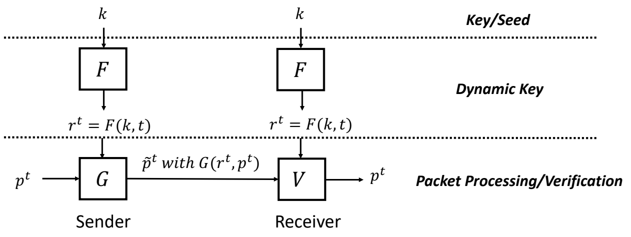


Fig. 1: SEAL's Framework

do not utilize authentication or explicit trust in PKI certificates in networking; instead of a certificate-authority-signed certificate, the nodes use self-certified keys for signing the packets used for the cryptocurrency application. There is no additional authentication mechanism beyond simply checking the identification in the networking header, such as the IP address and the port number, in blockchain networking. Our work focuses on securing the source integrity of networking via authentication so that we can distinguish an unauthorized attacker interjecting and injecting the communications after the connection has been established. In our SEAL scheme, only the key k has to be secret against any other nodes. We build SEAL on top of the TCP transport layer. We implement SEAL on the Bitcoin network for empirical analyses and facilitate practical deployment while considering the TCP networking and real-world deployment constraints.

B. Framework

We focus on the P2P pairwise connection between peer nodes i and j. Any node, excluding i and j, is considered unauthorized, and only nodes i and j possess the secure key k. The connection between i and j involves bidirectional communication, encompassing transmissions from i to j and vice versa. For the sake of clarity in our exposition, we concentrate on unidirectional communication. We assume two roles: one peer functions as the sender responsible for processing packets for authentication, while the other acts as the receiver to verify the received packets. The reverse direction follows the same authentication process. SEAL's framework is built on a symmetric cipher, where both the sender and receiver utilize the established key k, as depicted in the top row of Figure 1. As an additional security protection, both the sender and receiver employ a function F to generate a dynamic key r^t, using k and t (the sequential number of the packet sent from the sender). This process is illustrated in the middle row of Figure 1. Note that t is defined from the sender's perspective, posing a challenge on the receiver's side to accurately determine the correct t for verification purposes, which will be further discussed in Section V-D. After obtaining r^t, the sender locally takes p^t, the t-th packet sent from the sender to the receiver after session establishment, and generates on-the-fly packets $\tilde{p}^t$ by embedding a message $G(r^t, p^t)$ (computed based on the original packet p^t and the dynamic key r^t) into the original packet for authentication purposes. $\tilde{p}^t$ is then transmitted via the network. Upon receiving $\tilde{p}^t$, the receiver uses V to verify the packet by checking $G(r^t, p^t)$ and transforms $\tilde{p}^t$ back to p^t. This packet processing and verification stage is depicted in the bottom row of Figure 1.

C. SEAL Assumptions:

We list the assumptions for the SEAL framework in this section. These are standard assumptions in cryptography for cryptographic hash functions. We are therefore brief in our treatment of these assumptions, although we build on them for the SEAL design and analyses in the rest of the paper.

Assumption 1. *It is difficult to derive k from $r^t = F(k)$.*

Assumption 2. *Given $r^t, t \geq 1$, it is difficult to generate or guess $r^{t+n}, n > 1$.*

Assumption 3. *It is difficult to derive r^t from $G(r^t, p^t)$.*

Assumption 4. *Given $G(r^t, p^t)$, it is difficult to generate x and y such that $x \neq r^t$, $y \neq p^t$, $G(x, y) = G(r^t, p^t)$.*

Assumption 1 and Assumption 2 are related to the F design to prevent attackers deriving k or guessing the future r^{t+n} that can be used to hijack or inject packets to the connection. Assumption 3 prevents attackers from deriving r^t and re-using r^t to generate a malicious packet that will be accepted by the receiver. Assumption 4 about G corresponds to the weak-collision resistance property.

D. SEAL Application Scope

Cryptographic Primitives and Building Blocks: SEAL builds on the standard cryptographic primitives, which are believed to be robust/secure and widely used for our modern-day digital security. For example, we build on the cryptographic hash function (SHA-256 in our implementation) for F and G; more specifically, we build on the preimage resistance and the second preimage resistance/weak collision resistance properties stated in the Assumptions in Section IV-C. SEAL security holds if the security for the underlying hash function holds.

End-to-End: SEAL provides end-to-end authentication, in contrast to the authentication for broadcasting, e.g., TESLA [10]. Because SEAL is implemented on the packet header (more specifically, the TCP options field) without incurring the changes in the TCP protocol, SEAL does not interfere with the intermediate network node operations, such as the stateful firewalls using the other header fields, including sequence number, acknowledgment number, checksum, and flags, for detecting TCP anomalies.

V. SEAL: Secure and Efficient Authentication using Linkage

SEAL leverages the linkage between packets to establish end-to-end connection authentication for permissionless blockchain networking. The SEAL design aligns with the framework proposed in Figure 1 to mitigate the threats demonstrated in Section III, while also addressing the efficiency concerns outlined in Section II. Specifically, the SEAL design comprises the F function, responsible for generating the secure pseudo-random number r^t, and the G function, which computes a

Authorized licensed use limited to: Università di Bari Aldo Moro. Downloaded on September 17,2025 at 10:46:41 UTC from IEEE Xplore. Restrictions apply.

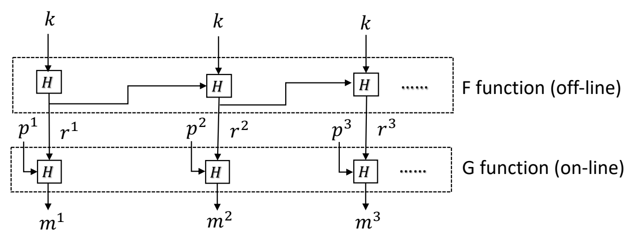


Fig. 2: The SEAL Design

secure message embedded in the packet header to ensure packet and source integrity. From a security perspective, adherence to all SEAL assumptions outlined in Section IV-C is crucial. Considering efficiency, it is essential to ensure that F operates offline for the in-advance construction of the pseudo-random chain. Conversely, since G and V operate online (computing when packets are sent and received, respectively), their operations must be carefully designed to ensure efficiency. In addition to security and efficiency considerations, the design of V must also address TCP networking constraints, such as out-of-order packets or packet loss, and real-world deployment constraints, including firewalls along the path between pairs of nodes for detecting TCP anomalies.

The SEAL design is depicted in Fig. 2. The upper rectangle represents the F function, which functions as a Pseudo-Random Number Generator (PRNG) for producing r^t and is inspired by the Output Feedback (OFB) mode [11]. This OFB-like design facilitates the generation of pseudo-random numbers using the established key, enabling in-advance construction (i.e., r^t can be computed offline). The lower rectangle corresponds to the G function, responsible for generating messages (denoted as m^t) to be embedded into the selected field in the TCP header for authentication purposes. In the subsequent discussion, we delve into the detailed functionalities of the F and G functions, as well as the verification process.

A. Key establishment in SEAL

Before communication and packet transfer, we employ the traditional prime-based Diffie-Hellman key exchange during the TCP three-way handshake to establish the symmetric key k for SEAL between the two peers. Analyses and comparisons of variations, such as Elliptic-curve Diffie–Hellman (ECDH), are left for future research directions. As discussed in the introduction, an attacker can generate a fake identity (spoofing) in permissionless systems and link it to the victim. Since the initial trust establishes the trust level, SEAL relies on the trust initially established. Such a model is related to Trust On First Use (TOFU) [12], [13] or the Resurrecting Duckling model [14], which postulates that the security is only as strong as what has been initially established. With such a model, if the spoofing attacker using a i's identity builds the initial trust, it is impossible to distinguish i and the attacker masquerading as i. Therefore, SEAL's security requires the prerequisite that the man-in-the-middle (MITM) attacker is absent at the key establishment and thus ensures a secure Diffie-Hellman key exchange (known to be vulnerable to the MITM attack). The threats on the key exchange against the key-exchange channel availability or the key integrity are beyond the scope of this work. The connection establishments are also distributed and P2P, as opposed to having a centralized vulnerability location in the network, limiting the threat feasibility scope against the key exchange.

B. The F function in SEAL

After the key establishment, the design of F for SEAL can be treated as a Pseudo-Random Number Generator (PRNG) motivated by OFB [11], known as a standard approach to building a cryptographically secure PRNG [15]. Such designs enable stream-cipher-like operations and in-advance construction of PRNG. The input of F is the established key k. The goal of the F function is to generate the secure inputs (r^t) for G. The H function used in F requires the pre-image and collision resistance property. More detail for this prerequisite will be discussed in Section V-E.

C. The G function for Packet Processing in SEAL

In the design of the G function, the sender uses the original packet p^t and the security input r^t to generate the message m^t for authentication purposes. The challenge lies in determining which part of the TCP header should be used to accommodate this message m^t. Initially, we consider the header field, such as the TCP sequence number, similar to the SYN cookie technique [16], which counters SYN flooding by encoding only the initial sequence number. However, modifying the sequence number field might lead to stateful firewalls dropping packets due to sequence-number checks enabled by major firewall vendors like Cisco, Juniper, and Check Point [17].

As outlined in Section IV-D, stateful firewalls typically monitor TCP sequence numbers, acknowledgment numbers, checksums, etc. However, the TCP options field is often exempt from monitoring because many options are only used during the initial SYN and SYN/ACK phases of the three-way handshake. By default, the TCP field of packets after the three-way handshake is always empty. This is why we choose to embed the message m^t into the option field of the original packet. While option fields can serve various purposes, such as Multi-path TCP, resulting in a non-empty option field after handshaking, we assume, for simplicity, that the TCP field of packets after handshaking is always empty for the sake of our explanation in the following discussion. If the option field is non-empty after handshaking, an additional step would be required to blend the original option field data with the message m^t.

To generate m^t, we perform a hash computation by concatenating the secure inputs r^t and the packet p^t. This ensures that r^t precisely corresponds to the legitimate packet p^t (tight coupling of r^t with p^t), as expressed by $m^t = G(r^t, p^t) = H(r^t||p^t)$. If we were to directly use r^t or $H(r^t)$ as m^t and write it into the option field, an attacker could intercept the packet, replicate the same m^t, and craft another malicious packet that the receiver would accept, thereby compromising authentication. Concatenating r^t with p^t as the hash input mitigates such

Authorized licensed use limited to: Università di Bari Aldo Moro. Downloaded on September 17,2025 at 10:46:41 UTC from IEEE Xplore. Restrictions apply.

vulnerabilities. The H function employed in G necessitates pre-image and collision resistance properties, discussed in Section V-E. For our implementation, we use SHA-256 for H to ensure pre-image and collision resistance. Given that the maximum size of the TCP options field is 40 bytes, we utilize the entire hash output to achieve 256-bit security.

D. Packet Verification on the Receiver in SEAL

The verification challenge arises from the receiver not knowing the value of t (as it is defined from the sender's perspective). To verify, the receiver needs to determine the correct t and obtain the corresponding r^t. In an ideal network scenario with no packet loss, the verification process is straightforward due to the consistent use of r^t. Taking p^t as an example, if the receiver has already received packets p^b, where $1 \leq b \leq t-1$, the verification begins with the reception of packet $\tilde{p}^t$. Using r^t for verification, the receiver retrieves m^t from the option field of $\tilde{p}^t$ and checks whether m^t equals $H(r^t||p^t)$ using the correct r^t and the corresponding p^t obtained by removing m^t from $\tilde{p}^t$. In practical cases, packets may be lost and retransmitted for reliable transfer. Consequently, the receiver might mistakenly use an incorrect r^s (where $s \neq t$) for verification. Notably, each r^x can only be used once, and some r^x may never be used for verification due to packet loss. To address this, we employ a queue, denoted as Q_R, with a fixed size of n_R, to manage the pseudo-random numbers used for verification.

For setting n_R, we consider the following question. Assume that p^y is the next received packet on j side after p^x. Then, what is the highest possible difference between x and y? To answer it, we consider the maximum number of packets sent from i. We categorize the TCP packets into the data packet (p^t with $l^t > 0$) and ACK packet (p^t with $l^t = 0$). The size of data packets that can be sent is $min(w_i^c, w_j^r)$ where w_i^c is the congestion window determined by the congestion control algorithm on i and w_j^r is the receiver window notified by j. However, from j's perspective, j does not know w_i^c but knows w_j^r. In addition, j does not know the message types of the following packets. Hence, j can only use the minimum size of a Bitcoin message, i.e., 24 bytes, the Bitcoin message header with no payload. Therefore, the maximum number of data packets that can be sent is estimated a $\lfloor \frac{w_j^r}{24} \rfloor$. On the other hand, the number of ACK packets (u_j) is related to the number of data packets sent from j that have not been ACKed yet. This information is known by j. Consequently, we set the finalized n_R as $\lfloor \frac{w_j^r}{24} \rfloor + u_j$.

The receiver sequentially utilizes $r^x \in Q_R$ to identify the correct r^t for verification. If a matching r^t is found (i.e., there exists $r^x \in Q_R$ such that $m^t = H(r^x||p^t)$), the packet $\tilde{p}^t$ successfully passes verification, and the original packet p^t is forwarded to the TCP kernel. Concurrently, r^t is removed from Q_R, and the next subsequent number is added to the queue. Additionally, if a previously verified packet used r^s to pass verification, it is necessary to check whether p^t is the next data packet following p^s by comparing their sequence numbers. Specifically, if $s^t = s^s + l^s$ (where s^t and s^s are the sequence numbers of p^t and p^s respectively), then p^t is identified as the next data packet after p^s. In such cases, pseudo-random numbers located between r^s and r^t in Q_R are removed if $t - s > 1$. This is crucial as these numbers are used for ACK packets or retransmitted packets, which could otherwise flood Q_R and disrupt our verification process. Furthermore, it's important to note that retransmitted packets arriving later than p^s can also use these numbers. However, as the original packets of the retransmitted packet should have been processed earlier, we don't need to handle the out-of-order packet case extensively. The focus of the verification process is on selecting the correct r^t. If packets arrive out-of-order, they can still be processed by finding the respective r^t in Q_R. The reassembly of these packets will be addressed further in the TCP layer.

E. SEAL Analysis

In our threat model, we assume that an attacker targets an existing connection and aims to spoof as a victim and inject messages as the victim without knowing the initial secure key k. If the initial key k is compromised, the attacker can generate all r^t for G and make the packets the receiver will receive. We consider the worst-case scenario that an attacker can observe or infer any r^t. We need to ensure that k can not be inferred (Assumption 1). To this end, H in F should have the preimage resistance as we mentioned in Section V-C. By Kerckhoff's principle, we assume that the attacker knows the design of F. Since H has the preimage resistance property, an attacker can not infer the key k by observing any r^t and uses the key to compromise the entire pseudo-random number chain. Moreover, we need to ensure that the attacker can not use r^t to infer any following $r^{t+n}, n > 1$ (Assumption 2). Otherwise, the attacker can use such a vulnerability to pass the verification and successfully inject the message as the victim. With the OFB-like design, the attacker can not use r^t to generate the following r^{t+n} without knowing k. However, suppose H has no collision resistance property. In that case, the same values can be generated, i.e., $r^t = r^{t+n}, n > 1$, the attacker can utilize r^t and make the malicious packet p^{t+n} being accepted by the victim by setting $r^{t+n} = r^t$. We thus ensure that H has collision resistance property in our design. Similar reasoning can be applied to the security assumptions related to the G functions. The attacker can sniff the information of $m^t = G(r^t, p^t)$, however, the preimage resistance of H protects the input r^t (Assumption 3) and the collision resistance of H ensures that given $G(r^t, p^t)$, is difficult to generate x and y such that $x \neq r^t$, $y \neq p^t$, $G(x, y) = G(r^t, p^t)$ (Assumption 4). With the above analysis, SEAL is secure against the threats defined in our threat model.

VI. SEAL Evaluation

For the performance comparison, we focus on four different schemes: No Security (the original operation without security protection), SEAL, ChaCha20-Poly1305, and HMAC. We select ChaCha20-Poly1305 because it is the cipher suite proposed using in BIP 151 [18] and BIP 324 [19] (proposals

Authorized licensed use limited to: Università di Bari Aldo Moro. Downloaded on September 17,2025 at 10:46:41 UTC from IEEE Xplore. Restrictions apply.

for P2P communication encryption for the Bitcoin network, and BIP 324 is an extended version of BIP 151 with a new message structure). In fact, ChaCha20-Poly1305 is also used by TLS [20] because of the efficiency it provided. We also select HMAC (HMAC-SHA-256 as the secure keyed hash algorithm) because it is a popular MAC-based authentication technique known to be secure [21] and is widely used for the purpose of authentication. We select these schemes to highlight the efficiency of SEAL. Generally speaking, the additional security protection can undoubtedly add additional overhead, but the lightweight design of SEAL ensures its efficiency from both host and network perspectives. In contrast, the other two schemes yield significant overhead compared to our SEAL scheme. In this section, we first measure the additional overhead using different schemes on single pairwise communication in Section VI-A. After we get the result, we use it for the value-assignment of additional costs when we conduct the networking simulation experiments using a Bitcoin network simulator [22] in Section VI-B. Specifically, we use the simulator to capture the cascading effect of the additional overhead among different schemes on a Bitcoin network and analyze the impact on the block propagation delay.

A. Host-based Prototype Experiment

1) Experimental Setup: We prototype our SEAL scheme on a real-world Bitcoin implementation. To be more specific, we implemented two Bitcoin nodes on machines with Intel Xeon CPU E5-2650 v3 at 2.30 GHz machine equipped with 32 GB RAM. For measuring the additional operations in the packet processing (embedding the hash output into the TCP options field) on the sender side and packet verification (verifying the packets and transforming them back to the original packet) on the receiver side, we implement our experiments in Python by using the Scapy and NetfilterQueue libraries. NetfilterQueue can direct packets to a queue in userspace before sending them out or receiving them to the TCP kernel. Facilitated by NetfilterQueue, some additional processes can be done, such as dropping a packet if it does not pass the verification. Using Scapy, it is easy to write (or remove) data into the TCP options field of a packet before sending it out (or sending it into) the TCP kernel. In our prototyping, two Bitcoin nodes are connected. These two nodes will use our sender/receiver scripts to communicate with each other. Both Bitcoin nodes connect to the other ten peers with regular Bitcoin communication. Along with the performance results in the following sections, we include the information of 95% confidence interval, and all the results are measured among 10^4 packets.

2) Online Cost Analyses: The online cost represents the additional overhead spent on real-time processing in G and V. Precisely, the online cost is computed by summing up the processing times on the sender side and the receiver side for executing G and V functions. In Fig. 3, we show the online cost using different schemes. One can observe that the performance difference compared between No Security and SEAL cases is relatively small because of the lightweight verification operation of SEAL. Specifically, SEAL's online

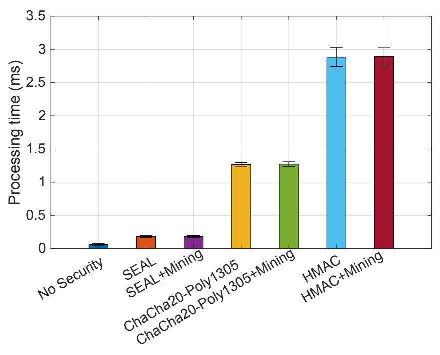
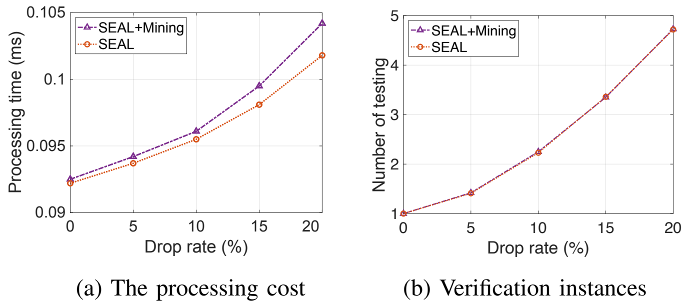


Fig. 3: Online cost comparison



(a) The processing cost (b) Verification instances

Fig. 4: The impact of varying drop rates

cost is 2.89 times bigger than No Security. However, the cost using SEAL significantly outperforms the online cost using ChaCha20-Poly1305 and HMAC. Specifically, the online cost of SEAL is 15.8 times smaller than the costs of HMAC (and 6.95 times smaller than the costs of ChaCha20-Poly1305). Furthermore, we also evaluate the SEAL, ChaCha20-Poly1305, and HMAC schemes with the miner mode enabled, indicating that the bitcoin node continues mining without interruption during operation. This assessment aims to determine whether the miner operation has any significant impact on processing time. The results indicate that the miner operation has a negligible effect on processing time for all the evaluated schemes.

3) Different Drop Rates: In the previous online cost analyses, we measured the cost under the ideal network environment where no packet loss happened. To measure the costs in a more realistic network environment and verify the feasibility of our verification algorithm, we manually control the drop rate on the sender side from 0% to 20%. In such a case, the receiver side's verification cannot always pass the examination on the first instance. In this regard, additional costs spent on verifying some more instances (r^x in Q_R) can be expected. We show the impacts on the processing time on the receiver while varying the drop rates in Fig. 4a. It reveals that even though the non-ideal networking can affect the performance, SEAL can still verify the packet under 0.1042 ms on average under the miner mode if the drop rate is less than or equal to 20%. We also measure the average number of verification instances, i.e., how many r^x are retrieved from Q_R for verifying a single packet, with different drop rates in Fig. 4b. Undoubtedly, it has a significantly positive correlation with processing time. Hence, the trend in Fig. 4a and Fig. 4b are quite similar.

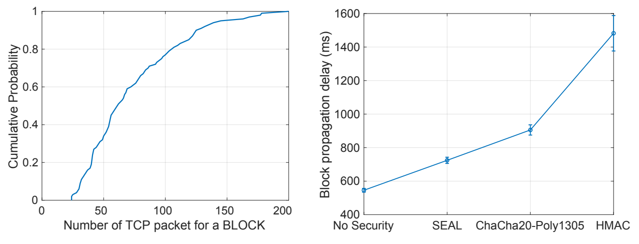
Authorized licensed use limited to: Università di Bari Aldo Moro. Downloaded on September 17,2025 at 10:46:41 UTC from IEEE Xplore. Restrictions apply.

B. Networking Simulation Experiment

1) Simulator Setup: To examine the impact of SEAL from a network perspective, we use a blockchain simulator [22] to obtain the block propagation delay using different schemes. To make it close to the real Bitcoin network, we assign the latencies based on the real-world data obtained from Bitnodes [23]. Specifically, we generate a random latency, l, for each link according to the cumulative distribution function (CDF) of latencies using the inverse CDF technique. When using different schemes, different overheads add up to the link delay. We treat it as an updated latency on each link so that the overheads using different schemes reflect the result of block propagation delay. Specifically, $l = l + c$ where c is the additional cost for sending a packet on a link using different schemes. We use the result obtained in Section VI-A2 to assign the value of c and get the updated l. The network scale is set to 12000 public nodes (i.e., the nodes accepting incoming connections and are the major components of the Bitcoin networks), each of which has 32 peer connections (i.e., the average number of peer connections of public nodes [24]).

2) Block Relaying Protocols: Different block relaying protocols affect the block propagation delay. To make it close to the real Bitcoin network, we follow the current Bitcoin operation built based on BIP 152 [25] with both legacy relaying and compact block relaying. Generally speaking, a node will send SENDCMPCT messages to at most three peers (provided the highest number of the recent N blocks the quickest) to initiate the compact block relaying. The remaining peers use the legacy relaying. Please refer to BIP 152 for more details. To simulate different relaying schemes, we set up a scale factor for the link latency to reflect the required number of packet transmissions using different schemes. Specifically, suppose a block propagation on a specific link requires k packets transmission. In that case, the link latency will be updated to $k \cdot l$. Different relaying protocols can yield different k. For the compact block relaying involving both high-bandwidth mode and low-bandwidth mode, k can be 1, 3, and 5. However, in our simulation, we randomly select $k = 1$ or 3. This is because public nodes almost always have all required transactions to rebuild the block mentioned in [26]. Thus, we exclude the $k = 5$ case where a node sends GETBLOCKTXN to request the required transactions to rebuild the block.

The remaining peers still use the legacy relaying operation for block transmission. In legacy relaying, k can be a significant number because of the transmission for a BLOCK message. Specifically, three bitcoin message transmissions are required for the legacy relaying, HEADERS or INV, GETDATA, and BLOCK. The payloads of the first two messages are small and can be encapsulated into one TCP packet. However, it requires more TCP packets for transmitting a BLOCK. To observe the required TCP packets (denoted as n_B) for transmitting a BLOCK message, we set up two Bitcoin nodes. A node (A) keeps updating to the latest block, whereas another node (B) starts updating the block after the difference in the block height reaches 100. We make sure that A and B are connected, and B will update the blockchain by receiving BLOCK messages from A. The CDF distribution of n_B is shown in Fig. 5a. We thus assign $k = 2 + n_B$ for the links using the legacy relaying where 2 is the number of packets for transmitting HEADERS or INV, and GETDATA; n_B is randomly generated according to the CDF distribution. The block propagation follows the shortest possible delay paths to reach every node in the Bitcoin network. We can expect that links using compact block relaying would highly affect the result of block propagation delay.



(a) CDF of n_B (b) Block propagation delay

Fig. 5: Simulation Results

3) Simulation Results using Different Schemes: We use the blockchain simulator [22] to simulate the delays when using different schemes. We show the impact of SEAL from a network view while comparing it with other schemes in Fig. 5b. We also provide the 95% confidence interval information in Fig. 5b. The result shows that the block propagation delay using SEAL is 1.25 and 2.04 times smaller than ChaCha20-Poly1305 and HMAC respectively. In other words, the cascading effect of the additional overheads using these two authentication schemes has a significant impact on block propagation performance. In other words, SEAL provides a more lightweight solution for bringing authentication into the permissionless blockchain network.

VII. Related Work

Since SEAL focuses on countering the networking-level threats based on TCP hijacking and spoofing, we mainly review those threats as follows. One Bitcoin-specific threat enabled by TCP spoofing is delay attack [2]. The delay attack is based on the BGP hijacking. Specifically, the authors propose utilizing the BGP hijacking to make the autonomous system an on-path network attacker and conduct partitioning and delay attacks. The attacker in the delay attack modifies the GETDATA message for requesting an older block while using the victim's identity (i.e., TCP spoofing). Such an attack can delay the latest block propagation for 20 minutes. A BiteCoin attack [27] is introduced by Recabarren and Carbunar to steal payments from the victim miner by performing TCP hijacking on the connection between the miner and pool server. There are two threats using both TCP hijacking and spoofing called ConMan (Connection Manipulation) attack [3] and Defamation attack [4], [5]. For the ConMan attack, an attacker tries to eclipse a target node's peer connection which is similar to the Eclipse attack [28] and EREBUS attack [29]) but using a connection manipulation approach (i.e., hijacking the connection based on spoofing.)

Authorized licensed use limited to: Università di Bari Aldo Moro. Downloaded on September 17,2025 at 10:46:41 UTC from IEEE Xplore. Restrictions apply.

In [4], [5], the authors present the Defamation attack. Under the Post-connection Defamation attack case, the attacker must hijack the connection between i and j and send misbehaving messages to a node i using j's identity (spoofing) and makes i ban j for 24 hours. The above Bitcoin-specific threats motivate us to design SEAL to provide authentication for the connection and defend against those threats.

VIII. Conclusion

In this work, we highlight the critical absence of cryptographic protection in permissionless blockchain networks, leaving them vulnerable to threats such as spoofing and hijacking. We address this security gap by introducing SEAL. SEAL's lightweight design ensures high performance, with results showing that the block propagation delay using SEAL is 1.25 and 2.04 times smaller than ChaCha20-Poly1305 and HMAC, respectively, in our simulations. SEAL provides an efficient and secure solution for communication among permissionless blockchain peers, promising enhanced security and efficiency.

Acknowledgment

This research was supported by the National Science Foundation under Grant No. 1922410 and by Colorado State Bill 18-086. The work also received support from XJTLU Research Development Funding RDF-21-02-012 and XJTLU Teaching Development Funding TDF21/22-R24-177. Additionally, it was partially supported by the XJTLU AI University Research Centre, the Jiangsu Province Engineering Research Centre of Data Science and Cognitive Computation at XJTLU, and the SIP AI Innovation Platform (YZCXPT2022103).

References

[1] S. Nakamoto *et al.*, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[2] M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking bitcoin: Routing attacks on cryptocurrencies," in *2017 IEEE Symposium on Security and Privacy (S&P)*, 2017, pp. 375–392.

[3] W. Fan, S.-Y. Chang, X. Zhou, and S. Xu, "Conman: A connection manipulation-based attack against bitcoin networking," in *2021 IEEE Conference on Communications and Network Security (CNS)*, 2021, pp. 101–109.

[4] W. Fan, H.-J. Hong, S. Wuthier, X. Zhou, Y. Bai, and S.-Y. Chang, "Security analyses of misbehavior tracking in bitcoin network," in *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2021, pp. 1–3.

[5] W. Fan, S. Wuthier, H.-J. Hong, X. Zhou, Y. Bai, and S.-Y. Chang, "The security investigation of ban score and misbehavior tracking in bitcoin network," in *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*, 2022, pp. 191–201.

[6] U. W. Chohan, "The double spending problem and cryptocurrencies," *Available at SSRN 3090174*, 2017.

[7] N. Varshney, "Why proof-of-work isn't suitable for small cryptocurrencies," *Hard Fork. Retrieved*, pp. 05–25, 2018.

[8] L. Dashjr, "Bitcoin core nodes," https://luke.dashjr.org/programs/bitcoin/files/charts/software.html, 2023.

[9] A. K. Miller, J. Litton, A. Pachulski, N. Gupta, D. Levin, N. Spring, and B. Bhattacharjee, "Discovering bitcoin's public topology and influential nodes," 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:15600193

[10] A. Perrig, R. Canetti, D. Song, and J. Tygar, "Efficient and secure source authentication for multicast," in *NDSS*, 01 2001.

[11] H. M. Heys, "Analysis of the statistical cipher feedback mode of block ciphers," *IEEE Transactions on Computers*, vol. 52, no. 1, pp. 77–92, 2003.

[12] V. Dukhovni, "RFC 7435:opportunistic security: Some protection most of the time," 2014, https://tools.ietf.org/html/rfc7435#section-3.

[13] R. Shirey, "RFC 4949: Internet security glossary, version 2," 2007, https://tools.ietf.org/html/rfc4949.

[14] K. Watsen, M. Richardson, M. Pritikin, and T. Eckert, "RFC 8366: Ia voucher artifact for bootstrapping protocol," 2018, https://tools.ietf.org/html/rfc8366.

[15] A. Van Herrewege, V. van der Leest, A. Schaller, S. Katzenbeisser, and I. Verbauwhede, "Secure prng seeding on commercial off-the-shelf microcontrollers," in *Proceedings of the 3rd International Workshop on Trustworthy Embedded Devices*, 2013, p. 55–64.

[16] J. Lemon, "Resisting syn flood dos attacks with a syn cache," in *Proceedings of the BSD Conference 2002 on BSD Conference*, ser. BSDC'02. USA: USENIX Association, 2002, p. 10.

[17] Z. Qian and Z. M. Mao, "Off-path tcp sequence number inference attack - how firewall middleboxes reduce security," in *2012 IEEE Symposium on Security and Privacy*, 2012, pp. 347–361.

[18] J. Schnelli, "BIP151: Peer-to-peer communication encryption," March 2016, https://github.com/bitcoin/bips/blob/master/bip-0151.mediawiki.

[19] J. Schnelli, "BIP324: Version 2 peer-to-peer message transport protocol," March 2019, https://gist.github.com/jonasschnelli/c530ea8421b8d0e80c51486325587c52.

[20] A. Langley, W. Chang, N. Mavrogiannopoulos, J. Strombergson, and S. Josefsson, "RFC 7905: Chacha20-poly1305 cipher suites for transport layer security (tls)," 2016, https://www.hjp.at/doc/rfc/rfc7905.html.

[21] M. Bellare, "New proofs for nmac and hmac: Security without collision-resistance," in *Proceedings of the 26th Annual International Conference on Advances in Cryptology*, ser. CRYPTO'06. Berlin, Heidelberg: Springer-Verlag, 2006, p. 602–619.

[22] S. Wuthier and S.-Y. Chang, "Demo: Proof-of-work network simulator for blockchain and cryptocurrency research," in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, 2021, pp. 1098–1101.

[23] A. Yeow, "Bitnodes," https://bitnodes.io/.

[24] S. Wuthier, P. Chandramouli, X. Zhou, and S.-Y. Chang, "Greedy networking in cryptocurrency blockchain," in *ICT Systems Security and Privacy Protection*. Springer International Publishing, 2022, pp. 343–359.

[25] M. Corall, "BIP152: Compact block relay," April 2016, https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki.

[26] G. Naumenko, G. Maxwell, P. Wuille, A. Fedorova, and I. Beschastnikh, "Erlay: Efficient transaction relay for bitcoin," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 817–831.

[27] R. Recabarren and B. Carbunar, "Hardening stratum, the bitcoin pool mining protocol," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 3, pp. 57–74, 2017. [Online]. Available: https://doi.org/10.1515/popets-2017-0028

[28] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015.

[29] M. Tran, I. Choi, G. Moon, A. Vu, and M. Kang, "A stealthier partitioning attack against bitcoin peer-to-peer network," in *2020 IEEE Symposium on Security and Privacy (S&P)*, 05 2020, pp. 894–909.

Authorized licensed use limited to: Università di Bari Aldo Moro. Downloaded on September 17,2025 at 10:46:41 UTC from IEEE Xplore. Restrictions apply.