Version++ Protocol Demonstration for Cryptocurrency Blockchain Handshaking with Software Assurance

Arijet Sarker[+], Simeon Wuthier[+], Jinoh Kim[*], Jonghyun Kim[§], and Sang-Yoon Chang[+]
[+]University of Colorado Colorado Springs
[*]Texas A&M University-Commerce
[§]Electronics and Telecommunications Research Institute

***Abstract*—Cryptocurrency software implements the cryptocurrency operations. We design a software assurance scheme for cryptocurrency and advance the cryptocurrency handshaking protocol. More specifically, we focus on Bitcoin for implementation and integration and advance its Version-message based handshaking and thus call our scheme Version++. The Version++ protocol provides software assurance, which is distinguishable from the previous research because it is permissionless, distributed, and lightweight to fit its cryptocurrency application. Utilizing Merkle Tree for the verification efficiency, we implement and test Version++ on Bitcoin software and conduct experiments in an active Bitcoin node prototype connected to the Bitcoin Mainnet. This paper for the conference demonstration supplements our technical paper at CCNC 2023 for synergy but highlights the prototyping and demonstration components of our research.**

***Index Terms*—Cryptocurrency, Blockchain, Bitcoin, Software Assurance, Permissionless, Distributed, Merkle Tree**

I. Introduction

Bitcoin is a popular distributed and permissionless cryptocurrency with a market cap of 382.25 billion dollars as of October 2022. The potential peers in the Bitcoin network can exchange messages with each other after successfully establishing connections using the Bitcoin handshaking protocol based on the Version message. However, the current Bitcoin handshaking does not provide software assurance for the Bitcoin software (to verify whether the peer has the claimed software version or not). We design and build Version++ scheme [1] to include software assurance in such Bitcoin protocol using the standard cryptographic primitives and make it permissionless, distributed, and lightweight for Bitcoin peer-to-peer (P2P) networking compatibility. However, to attain these properties and enable its cryptocurrency application, the software assurance achieved by Version++ ensures that the prover (a Bitcoin peer) holds the correct software code files and has a tradeoff with remote code attestation [2], [3] which offers stronger integrity/security assurance including code execution. The Version++ protocol uses the Bitcoin software files and the Bitcoin peer's unique ID as inputs to compute the Proof using a Merkle tree data structure [4] to network with other potential Bitcoin peers for software code assurance of the respective software version. Our Merkle Tree-based approach prevents an attacker from reusing the Proof of other Bitcoin peers (due to the use of ID as an input) and from generating the correct Proof without the necessary code files.

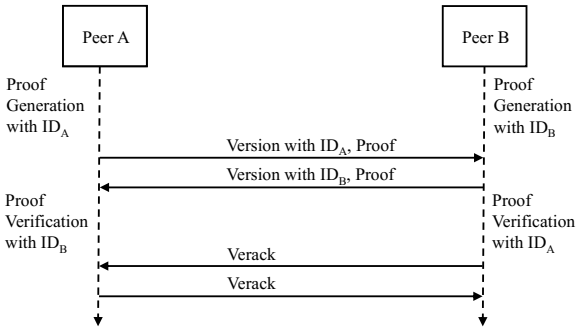
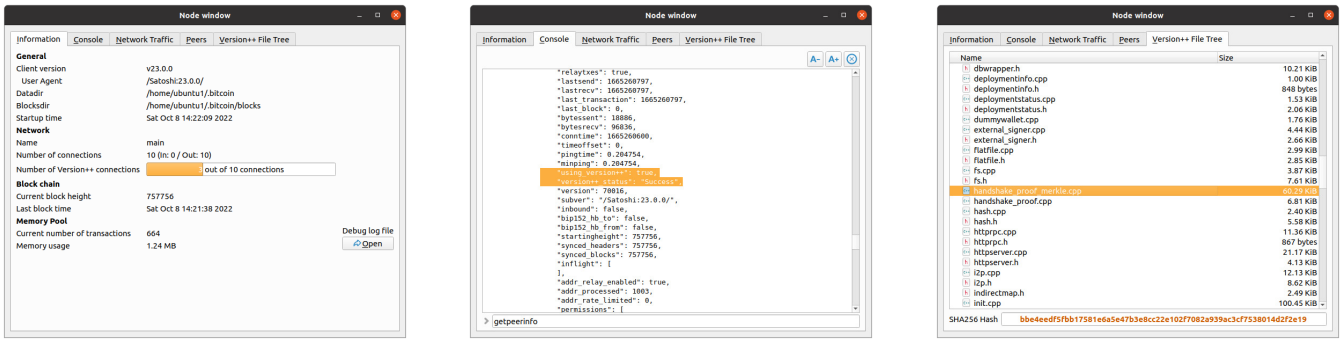


Fig. 1: Version++ handshaking built on the Bitcoin handshaking protocol. The only additions from the current handshaking protocol are the Version++ Proof in the Version message (networking) and the Proof verification (computing). The two peers interchange their roles as prover/verifier in handshaking.

In this demo paper, we describe the implementation details for the Version++ prototype involving a prover and a verifier in two virtual machines. We also connect a Bitcoin prototype to the Mainnet to measure the real-world P2P networking, including the peer connection duration and the current software version distributions providing us with references for the Version++ overheads. While this paper focuses on the implementation and demonstration, greater details and analyses of the Version++ protocol are in [1].

II. Version++ Scheme Protocol

Version++ handshaking protocol builds on the Bitcoin handshaking based on the Version and Verack exchanges and includes the Proof (a unique 32-byte data field for each peer because of the dependency on peer ID) for the software assurance as shown in Fig. 1. When two potential peers (e.g. peer A and peer B in Fig. 1) involve in the peer connection establishment process both peers become the prover (prove to other potential peers that it holds a specific Bitcoin software version) and verifier (verify whether its potential peer has the specific Bitcoin software version) to each other during the Version++ handshaking. In Version++, a prover uses the code files of its Bitcoin software version and the corresponding ID (e.g., IP address) as input of the Merkle Tree for Proof generation. The Merkle Root generated from the Merkle tree is communicated to the verifier as the Proof, and other



(a) Node information displyaing the number of connections supporting our scheme relative to all node connections.

(b) The node console showing *getpeerinfo* with "using_version++" and "version++_status".

(c) Our newly added tab, Verack++ File Tree, that enables exploring the files with each corresponding hash.

Fig. 2: Our Version++ implementation's user interface in Bitcoin Core.

hash nodes/outputs within the Merkle tree remain on the local machine of the prover. The verifier checks the Bitcoin software version number received from the Version message of the prover, recomputes the Proof for that version using the prover's ID, and compares the recomputed Proof with the Proof received from the prover. However, the verifier only needs to store the nodes/hash outputs of the Merkle tree that are affected by the ID and update those nodes with the prover's ID during the Proof verification. Both the peers send Verack message to each other only after successful verification and establishes connection with each other thereafter.

III. Implementation

We implement our Version++ prototype[1] in C++ and embed it into Bitcoin Core V0.23.0 (however, other Bitcoin implementations can be used). We make minimal protocol modifications to *net_processsing.cpp* (responsible for handling incoming and outgoing Bitcoin messages) and *net.h* (responsible for defining node connections). In *net_processing.cpp*, we add the Proof data field in the Version message arguments and call our Proof generation/verification functions when a Version message is received. In *net.h*, we store the Merkle Tree manager, *HandshakeProof* in the connection manager and add two flags in each node connection (*CNode* object): *isUsingHandshakeProof* flag and *handshakeProofStatus* to specify if the node is supporting the handshake Proof, and the status of the connection respectively. We also add a logging category to Bitcoin, "handshakeproof" that prints the status of each handshake to the console, including the Proofs being compared, and the state of the connection. For user interaction, we add an RPC call, *getversionproofhash*, and update the current *getpeerinfo* call to include our node state information, as shown in Fig. 2b, and update the user interface accordingly (Fig. 2a and Fig. 2c).

IV. Conference Demostration

Our demonstration is mostly software-based so we only require wireless connectivity to the Internet and an area for our laptop. During the demonstration, we will show our Bitcoin Core implementation and how it gets updated when other peers are using our Version++ scheme but resorts back to the default Version protocol when either of the nodes does not support Version++ (i.e. backward compatibility when Version++ cannot be performed). In Version++, when we modify any portion of the source code (thus changing the hash) the verification becomes unsuccessful (because the computed Proof and the received Proof from the prover does not match) and therefore the handshake fails. In the conference demo, we will highlight the functionalities of our interface and show how it affects the connection state within each peer connection.

[1]We make our implementation available at https://github.com/bitcoin-version-plus-plus/bitcoin-version-plus-plus.

V. Conclusion

We design and build a Version++ protocol for distributed, permissionless and lightweight solution of software assurance in a Bitcoin P2P network. Our Version++ protocol is generally applicable to the software-implemented permissionless and distributed cryptocurrencies in principle. However, in this paper, we demonstrate the functionalities and application behavior of Version++ on Bitcoin.

Acknowledgement

This work was supported by National Science Foundation under Grant No. 1922410 and by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2021-0-02107, Collaborative research on element Technologies for 6G Security-by-Design and standardization-based International cooperation). This demonstration paper supplements our main technical paper at CCNC 2023 [1] but highlights the prototyping and demonstration components of our research.

References

[1] A. Sarker, S. Wuthier, J. Kim, J. Kim, and S.-Y. Chang, "Version++: Cryptocurrency blockchain handshaking with software assurance," in *2023 IEEE 20th Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2023, pp. 1–6.

[2] M. Ammar, B. Crispo, I. De Oliveira Nunes, and G. Tsudik, "Delegated attestation: scalable remote attestation of commodity cps by blending proofs of execution with software attestation," in *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2021, pp. 37–47.

[3] I. D. O. Nunes, K. Eldefrawy, N. Rattanavipanon, and G. Tsudik, "{APEX}: A verified architecture for proofs of execution on remote devices under full software compromise," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 771–788.

[4] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Conference on the theory and application of cryptographic techniques*. Springer, 1987, pp. 369–378.