2023 26th International Conference on Computer Supported Cooperative Work in Design (CSCWD) | 979-8-3503-3168-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/CSCWD57460.2023.10152666

A Pure Hardware Design and Implementation on FPGA of WireGuard-based VPN Gateway

Jihong Liu*†‡, Neng Gao*‡✉, Chenyang Tu*‡, Yifei Zhang*‡, Yongjuan Sun*‡,
*State Key Laboratory of Information Security, Institute of Information Engineering, CAS, China
†School of Cyber Security, University of Chinese Academy of Sciences, China
‡Institute of Information Engineering, Chinese Academy of Sciences, China
Email: {liujihong, gaoneng, tuchenyang, zhangyifei, sunyongjuan}@iie.ac.cn

***Abstract*—In the face of rising dangers to the internal network due to remote cooperation, VPN gateways are an important tool for organisational network administrators, and the appropriate execution of VPN gateway functions is a vital component in safeguarding the internal network. The VPN gateway confronts security risks from the underlying cryptographic algorithm library, the current operating system, and the central processor as the number of attackers grows and attack methods evolve. In this paper, we propose a pure hardware logic VPN gateway to address security threats from the cryptographic algorithm library, operating system, and CPU by independently implementing the WireGuard protocol's underlying cryptographic algorithm and building the WireGuard protocol's hardware logic circuit on the FPGA platform. Actual testing on the NetFPGA-1G-CML platform reveals that the system's network throughput can reach 35Mbps/s, whereas the network throughput of the software's WireGuard VPN is 23Mbps/s under the same network settings. Simultaneously, the delay statistics of 300 repetitions of data packet encryption were performed. The encryption latency was less than 20 microseconds when the data packet size was the default MTU.098**

***Index Terms*—VPN Gateway, Remote Cooperation, FPGA, WireGuard, Pure hardware logic**

I. Introduction

Telecommuting has become a reality for many enterprises and organizations as a result of the COVID 2019 coronavirus epidemic. Because the majority of work nowadays must be done collaboratively by the organization's workers, telecommuting still needs a connection to the internal network for collaborative work. The necessity for comprehensive control over network traffic on internal networks clashes with the need to loosen firewall regulations to enable remote access, making it more difficult for network administrators to govern internal networks [1]. When employees of a company collaborate remotely, remote access solutions are required to assure data security, integrity, and availability throughout data transfer. As a result, network administrators frequently use VPN technology to secure communications during remote coworking, granting external hosts such as remote work devices access to internal resources [2]. A virtual private network (VPN) is an architecture that uses the tunnel technique to provide a logical safe channel for communication between two entities over the Internet [3]. VPN ensures data confidentiality, integrity, and availability using a variety of cryptographic techniques and security protocols. The weakness of the underlying cryptographic algorithm library is the first security issue found during VPN deployment. If a third-party encryption software library is used, it may be vulnerable to backdoor attacks, which are normally prevented by implementing the basic password algorithm independently. Second, while the security of VPN software code may be trusted, the VPN operational environment is equally dangerous. There will be a high number of vulnerabilities since the current operating system has a complicated structure and a large quantity of code. These flaws allow attackers to gain access to the system and execute arbitrary code [4], disrupting the regular operation of VPN operations. To avoid danger to the operating system, the trusted execution environment (TEE) will be employed to assure regular VPN function implementation. Third, the underlying hardware processor is assumed to be trusted in the traditional security assumption, but recent disclosures about attacks (Meltdown [5], Spectre [6], Foreshadow [7], and so on) that exploit performance-enhancing optimizations of the processor microarchitecture (e.g., out-of-order execution and speculative branch execution) are means by which this assumption is called into question, making even if the VPN is in a TEE environment, its functional execution remains a concern. To address the security risk posed by typical VPN gateways, we created a pure hardware VPN gateway that does not rely on a processor, operating system, or third-party code base. We select the WireGuard protocol developed by Jason Donenfeld et al [8]. Many researchers use this protocol because of its superior encryption technology and simple configuration. Many researchers have also demonstrated its security.We chose FPGA as the platform for implementing a pure hardware gateway. FPGA implements security functions through the use of finite-state machines or custom circuits. It lacks Turing machines' unlimited flexibility but has the virtue of considerably limiting or eliminating the potential of unintended execution. The method used to load the new configuration is designed to be segregated from the data stream through the FPGA, preventing the circuit from being modified during execution. Applying reconfigurable hardware to cyber infrastructure security has been a hot topic in the cybersecurity community [9]. FPGAs are a suitable and popular hardware platform for many cybersecurity applications, including protocol wrappers [10], packet classification [11], packet filtering [12], and intrusion detection [13]. We briefly discuss the

Authorized licensed use limited to: Malardalen University. Downloaded on June 18,2024 at 18:39:27 UTC from IEEE Xplore. Restrictions apply.

related work [14], [15]. The scheme in [14] provides an FPGA-based triple-DES implementation that greatly improves the throughput of DES and thus the performance of using Frees/WAN with IPsec. However, since only the optimization of the computational speed is provided by using FPGAs only, the data flow is ultimately controlled by the operating system. In Lu et al.'s scheme [15], IPSec is implemented by using a Xilinx Virtex-II Pro FPGA to separate the data path of IPsec from the control path, with key management and negotiation in the Power PC processor core. However, in this scheme, it is clear that key holding on the processor core is not secure enough. Most past research has focused on exploiting the superiority of FPGAs in cryptographic algorithm implementation to enhance the total processing rate of VPNs, hence boosting the entire system's communication efficiency. However, because the whole security device is still based on the CPU or the operating system, attackers may still tamper with the connection between the NIC and the FPGA, making the security function difficult to accomplish. This paper describes an FPGA-based pure hardware logic VPN gateway based on the WireGuard protocol. In our method, we use hardware circuits to construct the data processing plane of the WireGuard protocol core on the FPGA device, and the hardware code is developed autonomously to minimise the security issues associated with third-party code libraries. Network packets are received and processed directly by the hardware circuitry, which decouples the OS and CPU, preventing security concerns caused by OS and CPU vulnerabilities and backdoors. The devices in the LAN configure the VPN's control information using special packets to accomplish the separation of the control link from the data connection. In summary, the following are the important contributions of this paper:

1) We propose and design a VPN gateway that is entirely hardware code. To improve the security of the gateway system while it is running, we decouple it from the CPU, operating system, and third-party code base. We solve the problem that the security functions cannot be executed positively due to the security risks of the running environment when the traditional VPN gateway works.
2) We created a pure hardware logic gateway based on the WireGuard protocol on the NetFPGA platform. By modifying the WireGuard protocol implementation, we were able to reduce the effective logic code, making the entire hardware code more readable and auditable.
3) We evaluated and assessed the VPN gateway's security and performance. We evaluated the hardware VPN gateway against the software-implemented VPN gateway under identical network circumstances, and the hardware VPN gateway beat the software gateway in terms of throughput and encryption delay. Our tests demonstrate that hardware VPN gateways may be used in place of software gateways in actual situations.

II. Preliminary

A. WireGuard Protocol

WireGuard [8] is an open-source VPN protocol written in C by Jason Donenfeld et al. It is considered a next-generation VPN protocol, designed to solve many of the problems that plague other VPN protocols such as IPSec/IKEv2, OpenVPN, or L2TP. With its advanced encryption technology and simple configuration, it is favored by many researchers, who have also proven its security. WireGuard was originally developed for Linux but is now available for Windows, macOS, BSD, iOS, and Android. it is still in active development.

1) Cryptographic algorithm used by the protocol: WireGuard is a peer-to-peer communication protocol, and the level of each endpoint is the same. After a round of key agreement operations, the two endpoints will communicate. During the key negotiation, the two communicating parties play different roles as the communication initiator (Initiator) and the communication responder (Responder). WireGuard is highly modular [16]. The key exchange phase in the protocol called handshake and the subsequent data communication phase is completely separated. In the key exchange phase, WireGuard uses a variant of Trevor Perin's Noise [17] for a 1-RTT key exchange. WireGuard uses Curve25519 [18] for ECDH (Elliptic-curve Diffie-Hellman), HKDF (HMAC-based key derivation function) [19] for expansion of ECDH results, RFC7539 [20]'s construction of ChaCha20 [21] and Poly1305 [22] for authenticated encryption, and BLAKE2s [23] for hashing. Different from the key exchange using multiple algorithms, WireGuard uses only ChaCha20-Poly1305 specified by RFC7539 as the lightweight AEAD scheme in the data transmission phase.

2) Security of WireGuard: Since WireGurad's announcement, many people have verified its security. Donenfeld describes the security considerations of WireGuard design in [8], such as identity hiding and DoS attack mitigation. In addition to this, Donenfeld and Milner's symbolic model in [24] gives computer-verified proof of the WireGuard protocol. In [16], Dowling and Paterson present a computational proof of the WireGuard handshake that contains an additional key confirmation message. In [25], Peter Wu analyzes the security of WireGuard, considers that the protocol is relatively reliable, and suggests using the protocol when a secure network tunnel is required. Lipp and Blanchet et al. Made a mechanized cryptographic proof of the protocol in [26].

B. FPGA Platform

Field programmable gate array (FPGA) devices offer the flexibility of software and the high performance of hardware, making them a popular hardware platform for many network security applications, and the emergence of the NetFPGA platform demonstrates the high demand for incorporating FPGAs into network implementations. FPGAs are integrated circuits whose functionality is typically achieved by using hardware description languages (VHDL or Verilog) to specified. The vast majority of FPGAs on the market today are

Authorized licensed use limited to: Malardalen University. Downloaded on June 18,2024 at 18:39:27 UTC from IEEE Xplore. Restrictions apply.

based on volatile SRAM technology, which means that the functionality of the FPGA is specified by a configuration file in external non-volatile memory. Modern FPGA families support encrypted and authenticated configurations where the configuration file is never stored in plaintext in external non-volatile configuration memory and the FPGA decrypts and authenticates the contents of the configuration file when the FPGA is booted. The traditional use case for FPGAs in VPN gateway implementations is to reduce the burden on the host processor system by accelerating the performance-critical algorithms of the encryption protocol. Design techniques typically used to achieve acceleration include pipelining, parallelization, and cyclic unfolding, which can lead to excellent acceleration, as well as cost and power savings. However, the benefits of FPGA-based security implementations are not limited to performance gains; proper implementation of security features in FPGA logic can also improve the security level of the final product.

III. System Desig

In this section we first present our threat model, analyzing the vulnerabilities of existing systems where threats may occur. Then we present the system architecture and receive the functionality of each part of the hardware module separately. Finally, a security analysis is performed based on the system architecture as well as the threat model.

A. Threat Model

Several conditions are met in our hypothetical scenario. For starters, the attacker's physical location is outside the internal network, which means the attacker can only attack the organization's intranet through the internet. Second, the attacker cannot be fully simulated as an insider of the company, and the VPN should not consider the attacker's packets to be permitted through while it is functioning properly. Third, the FPGA device is trusted because it is completely under the network manager's control and the attacker has no physical access to the FPGA device. Typically, VPN gateways have a dual NIC topology, with the external NIC accessing the Internet through public IP and the internal NIC accessing the organization's LAN via private IP. After network packets are examined by firewalls, intrusion detection systems, and other security software, they reach the VPN module via the operating system, where the encryption protocol is executed before being delivered to the LAN via the internal NIC. There are various possible attack targets in the typical VPN gateway, including but not limited to:

- The operating system;
- Device drivers;
- Implementation of cryptographic primitives (for example, with vulnerabilities and backdoors in third-party encryption libraries);
- Compiler optimizations and possible microarchitectural changes between processor generations;
- The sheer depth of the software stack;
- Cache and memory management;
- Key management — for example, a buffer overflow bug can leak everything, including the secret keys;
- Lack of full control of the security algorithm implementation.

In addition to the items listed above, new investigations have called into doubt the security of the underlying processor architecture. Previously, the underlying hardware processor was assumed to be secure by default; however, recent disclosures about the possibility of performance-enhancing optimizations (e.g., out-of-order execution and speculative branch execution) using the processor microarchitecture (Meltdown, Spectre, Foreshadow, and so on) have called this fundamental assumption into question. Aside from security risks, typical VPN gateways may fail to fulfill needed performance (throughput and/or latency) and, in many cases, power consumption.

B. Hardware Module

We built three primary functional components in the hardware module design: the Peer information management module, the transmitting module, and the receiving module. These modules provide a one-way information link that completes the VPN gateway's security function.

1) Peer information management module: In FPGA, we will handle Peers by creating a Peer information table. As seen in the I, the Peer Information Table comprises each peer's public key, internal IP address, index pair, communication key pair, and authorized access IP address. Among these, the network administrator may only modify the Peer's public key, internal IP address, and the IP address authorized to be accessed through a specific packet, whilst the index and communication key pair are produced during the handshake.

TABLE I
Peer Information table

Public Key (32 bytes) 0x8520f0...4e6a	Sender (4 bytes) 0xaf56	Receiver (4 bytes) 0x0000
Sending Key (32 bytes) 0x000000...0000	Receiving Key (32 bytes) 0x000000...0000	Counter (4 bytes) 0x0000
IP (8 bytes)		0xC0A80001
Allowed IP		0xC0A8001c 0xC0A8001f ...

Public Key is the public key of this peer and is also the unique identifier of this peer. The *Sender* is the local index of the peer, and the *Receiver* represents the index of the peer on the other side, which is the *Receiver* in the communication packet. The *Counter* represents whether the connection is established or not, and the communication count and is used as a parameter of the AEAD algorithm during communication. *Public Key* and *Public Key* are the communication key pairs generated after negotiation with this peer, and the *Sending Key* is used to encrypt data when sending, and the *Receiving Key* is used to decrypt data when receiving.

Authorized licensed use limited to: Malardalen University. Downloaded on June 18,2024 at 18:39:27 UTC from IEEE Xplore. Restrictions apply.

In order to make it easier for network administrators to administer the VPN gateway, we built additional control data packets in addition to many data packets in the original WireGuard protocol that are specifically used to configure the VPN gateway. The II depicts the data bundle. This data package can only be transferred from the internal network to the VPN gateway for setup by the network administrator. Basic actions on Peer (add, delete, change, etc.) may be accomplished by altering the operation fields in the packet, as can activities such as fetching peer information and acquiring the FPGA device's public key.

TABLE II
PEER OPERATION PACKET

Type := 0x0 (1 byte)	reserved := 0^3 (3 bytes)
Index (4 bytes)	
Operation (4 bytes)	
Public Key (32 bytes)	
Internal IP (8 bytes)	
Allowed IP1 (8 bytes)	
Allowed IP2 (8 bytes)	
...	

- `Index`: The index of the peer.
- `Operation`: The operation the user wants to perform on the peer information table.
- `Public Key`: The peer's public key.
- `Internal IP`: The internal IP address assigned to this peer by the network administrator.
- `Allowed IP`: The internal IP address of the server that this peer is allowed to access.

The operation field indicates the operation on the peer information table. 0x0 means get FPGA public key, 0x1 means add peer, 0x2 means delete peer, 0x3 means to change peer information, 0x4 means to pull all peer information tables. When getting the FPGA public key, the *Index* is all zeroes.

2) Sender Module: This module accepts computer packets and transmits them to the Internet. The module has four primary functions: 1. Update and maintain the peer information table in accordance with the computer's peer operation packets; 2. Transfer a static private key to the computer; 3. Send a handshake message to a peer; 4. Encrypt and send communication packets from the machine to the network.

As shown in fig. 1, after the module receives the data packet from the computer, the processing flow of the module is as follows:

Step. 1: Determine if the packet is a WireGuard protocol packet, and discard the packet if it is not.

Step. 2: Check the contents of the data package and process the data package according to the *Type* in the data package. If *Type* = 0x0, go to **Step. 3**, and if *Type* = 0x4, go to **Step. 4**.

Step. 3: Processes according to the *Operation* in the packet, updates the peer information table, or sends information to the computer.

Step. 4: Retrieve the peer information table according to the index in the information package, and if a connection has been established with this peer, proceed to **Step. 6**, otherwise proceed to **Step. 5**.

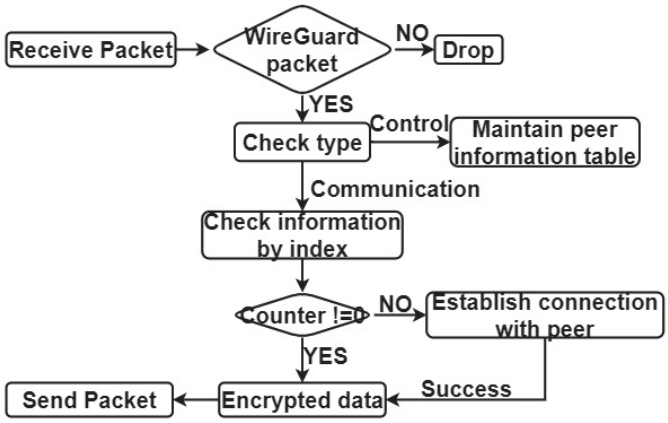

Fig. 1. Sender module flow chart

Step. 5: Send a handshake message to the corresponding Peer for key negotiation and proceed to **Step. 6** after success.

Step. 6: The data is first encrypted with ChaCha20-Poly1305 AEAD, and then WireGuard communication packets are constructed based on the data in the peer information table.

Step. 7: Send the encrypted data packet to the Internet.

3) Receiver Module: This module accepts Internet packets, processes them, and transmits them to the computer. The module serves two primary purposes: 1. Process the handshake information packet, complete the handshake to establish the connection, and then update the communication key in the peer information table; 2. Decrypt and transfer the communication packet received from the Internet to the computer.

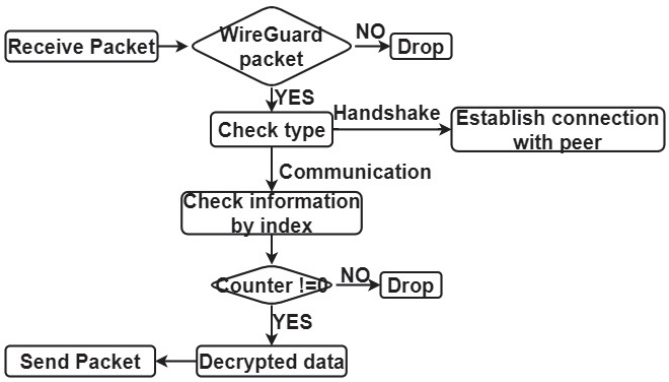

Fig. 2. Receiver module flow chart

As shown in fig. 2, when the module receives a packet from the Internet, the module's processing flow is as follows:

Step. 1: Determine whether the packet is a WireGuard protocol packet, if not, discard the packet.

Step. 2: Check the packet contents and process the packet separately according to the *Type* in the packet. If *Type* = 0x4, go to **Step. 4**, otherwise go to **Step. 3**.

Step. 3: Process the packet according to the *Type* to establish

Authorized licensed use limited to: Malardalen University. Downloaded on June 18,2024 at 18:39:27 UTC from IEEE Xplore. Restrictions apply.

the handshake connection.

Step. 4: Retrieve the peer information table based on the index in the information packet, and if a connection has been established with this peer, proceed to **Step. 5**, otherwise discard the packet.

Step. 5: Perform ChaCha20-Poly1305 AEAD decryption operation on the data.

Step. 6: Send the decrypted packet to the computer.

C. Security Analysis

Pure Hardware Logic Implementation. We built the VPN gateway utilizing immutable hardware logic code and a provably secure cryptographic protocol. Because network packets are received and processed directly by hardware circuitry, they are not vulnerable to the operating system, device drivers, processor architecture, or memory and buffer attack surfaces. Because the gateway function is implemented autonomously using pure hardware logic, it is totally divorced from third-party cryptographic libraries, operating systems, and CPUs, addressing the operational environment vulnerabilities that traditional VPNs face.

No Reliance On Third-party Code Base. For our gateway, we used the lightweight WireGuard protocol. We wrote the functional code in the solution as well as the cryptographic primitives in Verilog, giving us complete control over the VPN functionality while avoiding the risks associated with using third-party code libraries.

Simple Logic Implementation. Researchers admire the WireGuard protocol for its minimal weight, as less code implies fewer potential weaknesses. To prevent unknown states, we updated the WireGuard protocol so that the complete solution has fewer than 500 lines of functional logic code and leverages full-state machine jumps. As a result, the potential vulnerabilities in our solution are considerably minimised, ensuring the system's availability and stability. Furthermore, because of the limited quantity of hardware code and its implementation in modules, security auditing of hardware code is straightforward and possible.

IV. Implementation and Evaluation

In this section, we will introduce the specific implementation and test results of the VPN gateway.

A. Implementation Environment

In the solution implementation, we used NetFPGA-1G-CML [27] as the hardware deployment platform and the hardware code was developed according to the example architecture from Xilinx [28]. For testing, we also implemented the WireGuard software on a Linux 16.04 system and used it for network throughput comparison tests.

B. Componments

Each component is described and the resources required are listed. The main task of the Receive module (26149 LUTs, 5928 FFs, 113 DSPs) is to process packets from the Internet and send communication packets to the computer. This module contains cryptographic modules such as DH (Diffie-Hellman) module (13073 LUTs, 2708 FFs, 30 DSPs), AEAD module (5806 LUTs, 606 FFs, 83 DSPs), Hash module (3238 LUTs, 828 FFs) and KDF module (3724 LUTs, 964 FFs). The main task of the Send module (37391 LUTs, 8023 FFs, 143 DSPs) is to process the packets from the computer and to transmit the communication packets to the Internet. This module contains all the cryptographic modules in the Receive module, in addition to a Curve module (9530 LUTs, 1572 FFs, 30 DSPs) to generate temporary key pairs. The main task of the Peer Information Maintenance Module (838 LUTS, 268 FFs, 2 BRAMs) is to maintain the Peer Information Table.

C. Throughput Test

First of all, a network throughput test was conducted. We tested the throughput of the original WireGuard and our solution in the same network environment, and the results are shown in Fig. 3. When conducting throughput tests, the throughput of our hardware VPN gateway was stable at 35Mbps/sec, while the throughput of the software VPN gateway was stable at 23Mbps/sec. Instead of reducing the throughput of WireGuard itself, our solution has improved it. As for the factors affecting the throughput, we will discuss them in the next section.

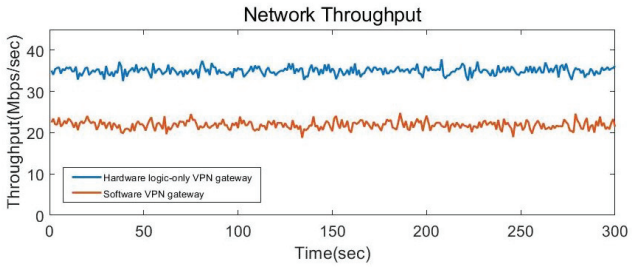

Fig. 3. 300-sec network throughput test results

D. Encryption Module Latency

We also tested the encryption latency of the hardware VPN gateway by collecting the latency data 300 times, and the results are shown in Fig. 4. The encryption latency of the hardware VPN gateway for network packets was less than 20 microseconds when the packets were the default MTU size.

V. Discussion

FPGA devices are a great way to create efficient cryptography algorithms [29]. The encryption technique used for the WireGuard protocol is simply implemented using Verilog code in our version, with no optimization. As a result, the code requires additional resources, and the number of concurrencies is insufficient. However, future hardware code optimization will undoubtedly reduce resource consumption in FPGAs, increase the number of concurrencies, and enhance communication efficiency. SoC FPGAs (for example, Xilinx's Zynq family [30]) integrate a CPU, a peripheral, and an FPGA into a single, programmable device. They offer more integration, reduced

Authorized licensed use limited to: Malardalen University. Downloaded on June 18,2024 at 18:39:27 UTC from IEEE Xplore. Restrictions apply.

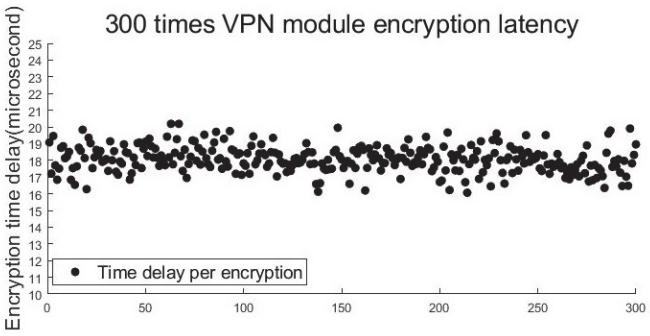

Fig. 4. 300 encryption time delays

power consumption, a smaller board footprint, and faster communication between the processor and FPGA. Some of the functionality underlying the VPN gateway can be implemented on the ARM chip of the SoC FPGA, resulting in a portable VPN security gateway device.

VI. Conclusion and Future Work

In this work we propose a hardware-only logical VPN gateway that does not rely on CPUs, operating systems, or third-party code libraries. The aim is to enable remote workers to work together more securely and efficiently. The lightweight WireGuard protocol is implemented autonomously using FPGAs to avoid vulnerabilities from third-party codebases, operating systems, and CPUs.

In the future, we will improve the encryption algorithm module in the FPGA to lower the algorithm module's resource consumption and increase the number of concurrencies, therefore boosting the VPN gateway's communication efficiency. Second, we will leverage the SoC FPGA platform to add authentication features in order to create a mobile VPN security gateway, providing personal remote usage security.

Acknowledgment

This work is supported by the National Key R&D Program of China (2022YFB3903900).

References

[1] F. Schwarz and C. Rossow, "Seng, the sgx-enforcing network gateway: Authorizing communication from shielded clients," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 753–770.

[2] M. Souppaya, K. Scarfone *et al.*, "Guide to enterprise telework, remote access, and bring your own device (byod) security," *NIST Special Publication*, vol. 800, p. 46, 2016.

[3] Z. Luo, G. Yu, H. Qi, and Y. Liu, "Research of a vpn secure networking model," in *Proceedings of 2013 2nd International Conference on Measurement, Information and Control*, vol. 1. IEEE, 2013, pp. 567–569.

[4] G. Sharma, A. Kumar, and V. Sharma, "Windows operating system vulnerabilities," *International Journal of Computing and Corporate Research*, vol. 1, no. 3, 2011.

[5] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin *et al.*, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 973–990.

[6] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher *et al.*, "Spectre attacks: Exploiting speculative execution," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 1–19.

[7] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient {Out-of-Order} execution," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 991–1008.

[8] J. A. Donenfeld, "Wireguard: Next generation kernel network tunnel." in *NDSS*, 2017.

[9] H. Chen, Y. Chen, and D. H. Summerville, "A survey on the application of fpgas for network infrastructure security," *IEEE Communications Surveys & Tutorials*, vol. 13, no. 4, pp. 541–561, 2010.

[10] F. Braun, J. Lockwood, and M. Waldvogel, "Protocol wrappers for layered network packet processing in reconfigurable hardware," *IEEE Micro*, vol. 22, no. 1, pp. 66–74, 2002.

[11] H. Song and J. W. Lockwood, "Efficient packet classification for network intrusion detection using fpga," in *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*, 2005, pp. 238–245.

[12] J. Loinig, J. Wolkerstorfer, and A. Szekely, *Packet filtering in gigabit networks using fpgas*. Citeseer, 2007.

[13] A. Das, D. Nguyen, J. Zambreno, G. Memik, and A. Choudhary, "An fpga-based network intrusion detection architecture," *IEEE Transactions on Information Forensics and Security*, vol. 3, no. 1, pp. 118–132, 2008.

[14] Y. H. O. Cheung, "Implementation of an fpga based accelerator for virtual private networks," in *IEEE International Conference on Field-programmable Technology*, 2002.

[15] J. Lu and J. Lockwood, "Ipsec implementation on xilinx virtex-ii pro fpga and its application," in *19th IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2005, pp. 7–pp.

[16] B. Dowling and K. G. Paterson, "A cryptographic analysis of the wireguard protocol," in *International Conference on Applied Cryptography and Network Security*. Springer, 2018, pp. 3–21.

[17] T. Perrin, "The noise protocol framework," *PowerPoint Presentation*, 2018.

[18] D. J. Bernstein, "Curve25519: new diffie-hellman speed records," in *International Workshop on Public Key Cryptography*. Springer, 2006, pp. 207–228.

[19] H. Krawczyk, "Cryptographic extraction and key derivation: The hkdf scheme," in *Annual Cryptology Conference*. Springer, 2010, pp. 631–648.

[20] Y. Nir and A. Langley, "Chacha20 and poly1305 for ietf protocols," *RFC 7539 (Informational), Internet Engineering Task Force*, 2015.

[21] D. J. Bernstein *et al.*, "Chacha, a variant of salsa20," in *Workshop record of SASC*, vol. 8, 2008, pp. 3–5.

[22] D. J. Bernstein, "The poly1305-aes message-authentication code," in *International workshop on fast software encryption*. Springer, 2005, pp. 32–49.

[23] J.-P. Aumasson, S. Neves, Z. Wilcox-O'Hearn, and C. Winnerlein, "Blake2: simpler, smaller, fast as md5," in *International Conference on Applied Cryptography and Network Security*. Springer, 2013, pp. 119–135.

[24] J. A. Donenfeld and K. Milner, "Formal verification of the wireguard protocol," Technical Report, Tech. Rep., 2017.

[25] P. Wu, "Analysis of the wireguard protocol," *Master's thesis. Eindhoven University of Technology, Eindhoven, Netherlands*, 2019.

[26] B. Lipp, B. Blanchet, and K. Bhargavan, "A mechanised cryptographic proof of the wireguard virtual private network protocol," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 231–246.

[27] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo, "Netfpga–an open platform for gigabit-rate network switching and routing," in *2007 IEEE International Conference on Microelectronic Systems Education (MSE'07)*. IEEE, 2007, pp. 160–161.

[28] I. LogiCORE, "Tri-mode ethernet mac v9.0 product guide, xilinx, 2018."

[29] W. N. Chelton and M. Benaissa, "Fast elliptic curve cryptography on fpga," *IEEE transactions on very large scale integration (VLSI) systems*, vol. 16, no. 2, pp. 198–205, 2008.

[30] X. Xilinx, "Zynq-7000 all programmable soc overview," *Product Specification, DS190 (v1. 10)(September 2016). accessed on*, vol. 9, 2017.

Authorized licensed use limited to: Malardalen University. Downloaded on June 18,2024 at 18:39:27 UTC from IEEE Xplore. Restrictions apply.