IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 22, 2020, accepted May 5, 2020, date of publication May 11, 2020, date of current version May 26, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2993700

Addressless: Enhancing IoT Server Security Using IPv6

RENJIE LIU, ZHE WENG, SHANSHAN HAO, DELIANG CHANG, CONGXIAO BAO, AND XING LI
Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China
Corresponding author: Xing Li (xing@cernet.edu.cn)

ABSTRACT Nowadays more and more IoT devices, including a large number of IoT servers, have been deployed on the Internet. The security of IoT servers has always been a challenge. In this paper, a new model named addressless IoT server is proposed, which allows people to use the large IPv6 address space to protect IoT server security. The server is allocated an IPv6 prefix instead of an address. When the authenticated client initiates communication, it uses an encryption mechanism to generate a specific destination address under the prefix. The server verifies the destination address when receiving the packet, and discards the packet if the verification fails. In this way, the model can prevent attackers from perceiving the server and launching scans or attacks, while remains compatible with the current Internet. The prototype is implemented and an extensive set of experiments are conducted in this paper. The results demonstrate that the model can better protect server security.

INDEX TERMS IPv6, IPv6 address space, Internet of Things, network security, prefix delegation.

I. INTRODUCTION

Starting from the birth of the Internet, the TCP/IP protocol has gradually become the most important infrastructure of the Internet, and the IPv4 protocol has been widely used after decades of development. However, the IPv4 addresses are being exhausted nowadays since it did not expect the huge number of devices on the Internet when the IPv4 protocol was designed.

To address this problem, the IPv6 protocol specification (RFC 1883 [1]) was proposed by the Internet Working Group. Now the latest IPv6 protocol is RFC 8200 [2] proposed in 2017. The most significant difference between IPv6 and IPv4 is that IPv6 uses a 128-bit address instead of the 32-bit address of IPv4, which provides a much larger address space for Internet devices. Besides, IPv6 also brings changes in mechanisms, such as using ND [3] instead of ARP on the access layer, adding a flow label [2] field in the IPv6 header, prohibiting intermediate router fragments, etc.

Now, IPv6 is becoming widely deployed for commercial purposes. As of Jan 2020, about 25% of global information resources support IPv6 (including web pages, Email, etc.), about 57% of global DNS authoritative servers support IPv6, and about 28% of users are using IPv6 to visit their target websites.[1]

Nowadays one of the most significant hotspots for IPv6 applications is Internet of Things (IoT) [4]. With the rapid development of IPv6 and 5G, IoT is increasingly changing people's life. According to Statista's forecast, by 2025, more than 75 billion IoT devices will be in use worldwide.[2] It is easy to observe that with the further development of the Internet, cloud computing, and the larger-scale deployment of the IoT, the number of IP addresses continue to increase, leading to a high-speed development of IPv6. However, IPv6 is still a developing technology with many open issues to be resolved. Specifically, the security of the IPv6 network for IoT remains a critical challenge.

Despite that for IoT, security is always one of the most concerned issues for network applications. Attacks have caused tremendous harm in the past decades and will continue to launch attacks to clients and servers, paralyzing devices, stealing information, or gaining benefits. For public Internet servers, service providers can deploy complicated security policies to prevent service suspension, secret theft, and information leakage caused by network attacks. However, for IoT servers, it poses more challenges to security because

The associate editor coordinating the review of this manuscript and approving it for publication was Chakchai So-In.

[1] https://www.vyncke.org/ipv6status/
[2] https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide

they need to provide uninterrupted and guaranteed Internet services to specific users. Therefore, protecting IoT server security is a very important topic.

In many network attacks, the IP address is used as the device identifier at network-layer to launch attacks. In IPv4, addresses are insufficient to protect the server using address space. However, in IPv6, the server can use a larger address space to hide the real addresses which provide services. For example, in IPv4, scanning a device by IP address is very easy. Scanning the entire IPv4 address space costs only 45 minutes using Zmap [5], whereas scanning the entire IPv6 address space is still impossible for now. This means that the large address space of IPv6 is conducive to server security. Although the progress of researches on IPv6 scanning in recent years makes it easier to scan the IPv6 servers, the IPv6's huge address space still has great potential to protect the server security.

The existing network security model usually encrypts the flow in the application layer or transport layer to protect security. IPsec [6] and the protocols based on it conduct encryption at the network layer, however, the entire packet payload is encrypted. Introducing encryption into the address is still a novel idea. It can further take advantage of IPv6's vast address space and introduces less additional load to the network traffic than encrypting the entire payload. Previous studies have suggested that IPv6 address space can be used to protect the security, but no specific model has ever been proposed. Using the IPv6 address suffix to hold the encryption information is an innovative idea, in accordance with the mainstream development trend of the IPv6 network model.

In this paper, a new model named addressless IoT server is proposed, which introduces encryption into IPv6 addresses in the network layer to enhance server security. The model is based on the Prefix Delegation and allocates an IPv6 prefix instead of an IPv6 address to a server. Then all addresses under the prefix will be listened on. Encryption based verification is used to enable different authenticated users to connect to different IPv6 addresses under the prefix. The server discards all data packets from unauthenticated users, to ensure that the server only responds to trusted data packets, reducing security risks faced by the server.

The rest of this paper will be organized as follows: Related work is introduced in Section II, and the design of addressless IoT servers is introduced in Section III; In Section IV, the security analysis on the model is proposed, and in Section V, the other design considerations and discussions of the addressless IoT server are introduced; The prototype implementation and the experiments based on our prototype are introduced in Section VI, and in Section VII, some future work is introduced. Section VIII is dedicated to the conclusion.

II. RELATED WORK

In this section, related work is introduced in the following three aspects: IoT security, IPv6 address security, and Prefix Delegation Model.

A. IoT SECURITY

IoT security is a complex topic. A large part of work is focused on how to modify the existing network security models in IoT scenarios in which network and computing resources are limited. Hwang [7] introduces some of the requirements of IoT security, including providing end-to-end security, providing security at different levels, easy to understand and configure, and considering resource constraints.

Previous researches are mainly focused on proposing lightweight network models and transmission protocols to reduce the resource consumption in the data transmission. 6LowPAN [8] proposed in 2007 is a wireless personal area network protocol, which is a short-range, low bit rate, low power, and low-cost protocol. Ghada Glissa *et al.* introduce 6LowPsec [9], which uses encryption under the MAC security sublayer by hardware to provide an end-to-end security solution. Raza *et al.* [10] propose an extension of 6LowPAN, which supports IPsec [6] in 6LowPAN to ensure the security of the network-layer. Oliveira *et al.* [11] propose a network access security framework that can be used to control the nodes which have access to the network to enhance the security for 6LowPAN.

The work based on the 6LowPAN protocol is mainly focused on the data link layer and the network layer. At the transport layer and application layer, IETF proposed the CoAP protocol [12] to reduce the transmission burden of the application layer by replacing TCP with lightweight UDP. On this basis, Shahid Raza *et al.* introduce Lithe [13], an integration of DTLS [14] and CoAP [12], with which the authors propose a novel DTLS header compression scheme to reduce the energy consumption by using 6LowPAN standard.

With the development of IoT, Fog networking [15] is proposed to integrate the computing resources of the IoT network better. In Fog computing, how to enhance the security in the low-resources edge devices is a big problem. Venckauskas *et al.* introduce SSATP [16], a lightweight protocol used as a secure transport for CoAP instead of UDP and DTLS for communications between Edge nodes and Fog nodes. Venckauskas *et al.* also introduce LSSP [17], a lightweight secure streaming protocol for the fog computing, which modifies UDP packets to embed authentication data into streaming data. Ghafoor *et al.* introduce XKFS [18], a lightweight scheme that allows refreshing the key without inter-node message transmission in WSNs, and Anand *et al.* introduce TARE [19], a scheme for lightweight and secure group communication, reducing the overhead and performance load in key management.

The works above are mainly focused on introducing a more lightweight network solution under the framework of the Internet of Things, which consumes fewer resources than the security protocols used in the traditional Internet. This paper creates a new method in a different aspect where encryption is introduced into the IPv6 address, and proposes a new model to ensure the security of the server in IoT scenarios. This is a new model which is different from the existing Internet encryption mechanisms, and can be used in various scenarios.

B. IPv6 ADDRESS SECURITY

The security problems brought by the IPv6 address space can be traced back to the early IPv6 protocols. There are several methods for IPv6 address allocation, including IPv6 Stateless Address Autoconfiguration (SLAAC) [20], DHCPv6 [21], and manual configuration. In RFC 4291 [22], IETF divides an IPv6 address into two parts, a 64-bit prefix and a 64-bit interface identification (IID). In SLAAC, the IID is calculated by the MAC address of the device through the EUI-64 algorithm [20]. EUI-64 is a reversible transformation, which means that anyone can calculate the MAC address through the IPv6 address of the user, thereby obtaining the device information of the user. This exacerbates the security and privacy problems.

Some approaches have been proposed to solve this problem. One of the most important methods is to generate semantically opaque IID with SLAAC [23] or DHCPv6 [24], [25]. SLAAC uses temporary addresses that consist of a fixed prefix and a periodical IID generated by the previous IID via the MD5 hash algorithm [26]. However, it brings difficulties to network management and may be faced with certain threatens in some circumstances [27]. Some other methods are introduced in RFC 7707 [28] and RFC 7721 [29], such as reducing the predictability of addresses or changing the address if necessary. RFC 7707 [28] concludes some IPv6 scanning tools and hitlist generation methods.

Just like those mentioned in RFC 7707 [28], IPv6 scanning is an important issue in IPv6 address security research. Zmap [5] greatly improves the scanning efficiency of IPv4, making it possible to scan the entire IPv4 address space within one hour. Similar to Zmap, Zmapv6 [30] is introduced in IPv6 scanning. However, IPv6 has a very large address space, so scanning the entire IPv6 address space is rather difficult. Some recent IPv6 scanning algorithms are based on the comprehensive description of IPv6 address space. A lot of works have been proposed to tell us how people use the IPv6 address in the past few years. Plonka and Berger [31] describe the temporal and spatial characteristics of active IPv6 addresses, and Li *et al.* [32] show us the prefix distribution of IPv6 Internet.

There are two types of IPv6 scanning methods, one is to collect the active addresses on the Internet, and the other is to generate the hitlists by learning and predicting algorithms. Fiebig *et al.* [33] use DNS data to collect active IPv6 addresses, while DNSSEC reverse zone is used by Borgolte *et al.* [34] and reverse DNS information is used by Fiebig *et al.* [35]. Beverly *et al.* [36] collect router addresses by some algorithms, such as random detection, and Rohrer *et al.* [37] accomplish some other similar work. Gasser *et al.* [38] summarize these methods and give a large hitlist set.

Another category of scanning method is to generate the hitlists by learning and predicting algorithms. Foremski *et al.* [39] first give a prediction method based on the Bayesian method, and Ullrich *et al.* [40] introduce a pattern-based scanning approach. Zuo *et al.* [41] try to do some prediction on association rule learning. Another important work is done by Murdock *et al.* [42], they generate a hitlist by some active seeds. Liu *et al.* [43] introduce 6Tree, a method on hierarchical clustering on the active IPv6 addresses prediction. These works make IPv6 scanning no more an impossible task, however, there is still a lot of work to do on this topic. Meanwhile, some other works are proposed to improve privacy and security on the IPv6 network. Fukuda and Heidemann [44] try to detect IPv6 scanning and evaluate relevant severity. Plonka and Berger [45] introduce a new method to increase the anonymity of IPv6 addresses.

Besides, there are some studies to protect user security and privacy using the huge address space of IPv6. Aljosha *et al.* [46] use an address hopping algorithm to change the address of the IoT server by a pre-communicated address generation algorithm to enhance the security of the IoT server and reduce the risk of being scanned. Dunlop *et al.* [47] introduce an idea that repeatedly rotating the addresses of both the sender and receiver to maintain user privacy and protect against targeted network attacks.

C. PREFIX DELEGATION MODEL

In IPv4, there is no prefix delegation. IPv4 addresses are configured by DHCP or static configuration. IPv6 allows to delegate a prefix to a device for some networking reasons. This device can allocate the addresses under the prefix to other devices after being allocated the prefix. DHCP-PD [24], [48] is a typical example. It allows the DHCP server to assign a prefix to a DHCP client, and the DHCP client can act as a DHCP server to assign the addresses under the prefix to other hosts in the subnet.

Besides DHCP-PD, researchers propose more work about prefix delegation. RFC 8273 [49] proposes an approach to allow a host to be configured a unique IPv6 prefix. This RFC allows each client to be configured an IPv6 address with a unique prefix to ensure that hosts cannot send packets to each other except through the first-hop router. As a result, it can provide isolation between the connected visitor devices in a shared-access network. RFC 8273 [49] proposes the idea of configuring the hosts with the addresses of different prefixes, but it does not make full use of the huge address space under the prefix.

III. DESIGN OF ADDRESSLESS IoT SERVER

In this section, the mechanism of addressless IoT Server is introduced.

A. DESIGN PRINCIPLES

The model proposed in this paper is designed to ensure server security through massive IPv6 address space. To reach this goal, the model takes advantages of the following three features:

1) Introduce encryption into IPv6 addresses, by which the redundant space of the IPv6 address can be used.

2) Eliminate the one-to-one correspondence between the server and the IP address.
3) Use Prefix Delegation model to assign an IPv6 prefix instead of an IPv6 address to each server, so that the server can use a segment of IPv6 addresses.

In this way, the server can use all of the addresses under the prefix, which means a huge amount of addresses can be used to hide the address that provides service. It also means that the server can use different addresses to provide services to different users.

In the model, ''addressless'' means that the server doesn't have a fixed IPv6 address. It is configured with a prefix instead. While it communicates with a client, it uses a temporary address in a connection. This address is generated by an encryption-based algorithm in an authenticated client. The server cannot be perceived by unauthenticated devices. So, ''addressless'' here also refers to a state that is not visible to the outside world.

B. SYSTEM DESIGN

As presented in Section I, the addressless server is a private server that only provides services to authenticated Internet clients. It should be noticed that the authenticated client is just a common Internet client with an IPv6 address.

The topology of an addressless IoT server is shown in Fig.1. The server is connected to the first-hop router. The server and the router are both configured with non-public IPv6 addresses, usually link-local addresses for network configuration. The first-hop router allocates the prefix and configures the relative route to the server, so that all the packets whose destination addresses are under the prefix can reach the server. Theoretically, the server can also be configured with a non-global unicast address, such as a ULA [50] address. For security reasons, this address should be invisible to the outside network.

When a client gets authentication from the server, a pair of encryption keys are generated and saved separately in the server and the client. This authentication can be done when initializing a newly purchased camera or configuring a health bracelet, etc. This process is done before the first communication. The key exchange process is not the focus of this paper, so we assume that it is secure and trustworthy.

When the addressless server starts its work, it listens on all the IPv6 addresses under the prefix. When an authenticated client initiates communication with an addressless server, the client uses its own IPv6 address as the source address first, then generates an IPv6 suffix through the source address and the encryption key. Then the client combines the server prefix and the suffix to form an IPv6 address as the destination address. Finally, the client sends the packets to the server.

The source address is described as SA, the encryption process is described as function f, the prefix of the server is described as $prefix$, and the prefix length is described as N, then the destination address DA is:

$$DA_{1:N} = prefix \tag{1}$$

$$DA_{N+1:128} = f(SA) \tag{2}$$

The specific process of f is described in Section III.D.

When the server receives a data message, it uses the decryption key saved in the server to verify whether the packet is from an authenticated client. The server extracts the source address and the destination address of the packet, take the source address as plaintext, and the suffix of the destination address as ciphertext to make the verification. If the packet is legitimate, that is, the packet is from an authenticated client, then the packet will be sent to the operating system or appropriate application. Otherwise, it will be dropped.

The decryption process could be described as (3):

$$Result = g(SA, DA_{N+1:128})) \tag{3}$$

In (3), $Result$ is a bool value stands for the verification result. True means the packet is from an authenticated user, and vice versa. $g()$ is the verification function.

There are two things to be noticed:

1) The client is just a common IPv6 client, configured an IPv6 address, not a prefix.
2) The encryption-decryption process described above is done once in a flow. That is, when the client initiates a connection, it generates a destination address by the algorithm. After a flow is established, there is no need to generate a new destination address until the connection is terminated. And at the same time, the server makes the verification only during the connection establishment. After the connection is established, the server accepts the packets directly.

If a connection is terminated, and the client intends to re-initiate a connection to the server, it calculates a new destination address. The server should conduct the verification for the new connection as well. However, UDP and ICMP are not connection-oriented protocols, but considering that UDP and ICMP also have the concept of data flow, the verification at the flow level is easy to perform as well. Determining whether different UDP or ICMP packets are in the same flow has been well implemented in the operating system or network devices, such as Linux or Cisco NetFlow [51], so we do not discuss it here.

It is important that when the connection terminates, the destination addresses generated by the same client in the next connection should be different. Otherwise, a client always uses the same destination address to initiate communication with the server. A man-in-the-middle can easily learn the ''source address-destination address pair'' to launch replay attacks. This will harm the security. Therefore, we should consider adding time-varying parameters in the encryption process.

How to select the time-varying factor will be discussed III.E.

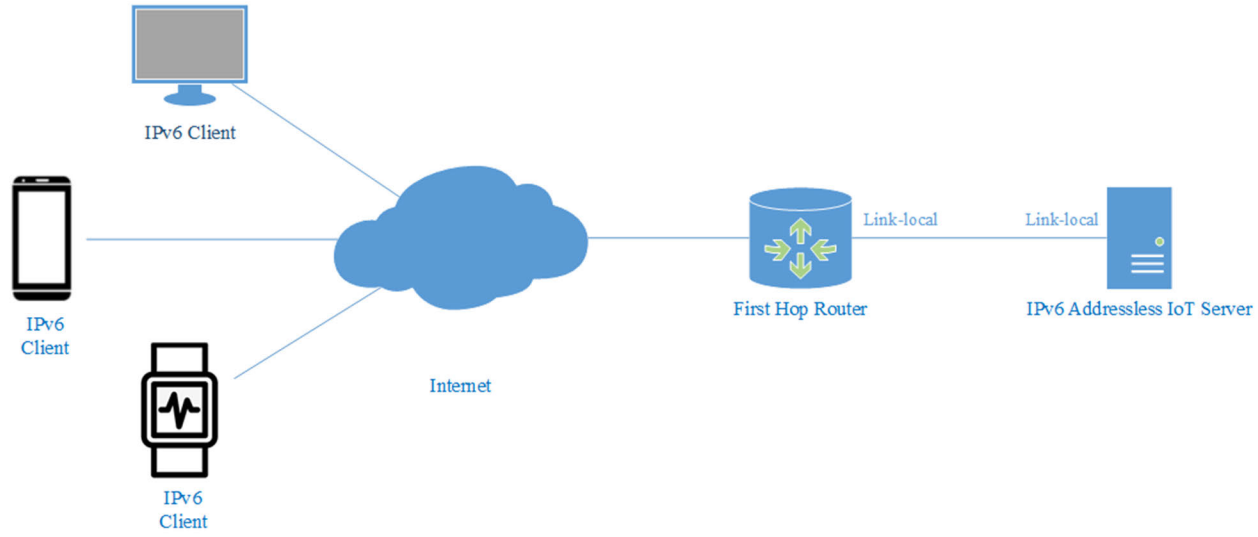


FIGURE 1. Topology of addressless IoT server. The server is connected to the first-hop router, through which connected to internet.

In the discussion above, the encryption is conducted at the flow level. However, encryption at the packet level is also feasible. In this case, each packet generated by the authenticated client has a different destination address, and each packet received by the server will be verified. This brings some additional security features, which will be discussed in Section IV. However, it increases the resource consumption of the server, and brings bigger challenges in the case of network jitter or retransmissions. As a result, encryption at the flow level is a better choice.

From the above discussion, it can be seen that our model takes advantage of the redundancy of the IPv6 address space. We allocate a prefix to a server, free up the address space of IPv6 suffix, and let it carry the authentication information to conduct the verification at the server-side. In our model, the source address and the prefix of the destination address are used in routing. We attach the verification information in the suffix space of the destination address to provide additional security attributes. In the previous network communication model, the authentication is generally performed at the transmission or application layer, such as HTTPs [52]. In our model, we conduct verification in the network layer. Authentication and encryption at the upper layer is fully compatible with our model, we can use it with our model at the same time to further enhance the security.

C. PREFIX LENGTH CONSIDERATION

The length of the prefix is affected by the requirement of routing. Generally, the prefix used for routing is no longer than /64, because a traditional IPv6 address includes a /64 prefix and a /64 interface identifier (IID) as suggested in RFC 4291 [22] and related RFCs. The prefix is used for routing and the IID is used for identifying interfaces on a link. Therefore, a /64 prefix is a good choice. However, this is not mandatory. If the subnet has a short prefix, and there are few servers in the subnet, to allocate a /56 prefix or even a /48 prefix is reasonable. And in some special scenarios, for example, a family is assigned a /64 prefix, but there are several IoT servers that need to be allocated a prefix. In this case, a /68 prefix or a /72 prefix is reasonable as well.

The length of the suffix is affected by the requirement of verification. The suffix space is essentially the ciphertext space of the authentication information. A longer suffix length means there is more space to increase the encryption security level. As a result, we need to guarantee that the suffix length is not too short. An extreme example is that if the prefix is /120, an attacker can traverse it within 256 tries. It is obviously very insecure. If an attacker can easily find the solution through brute force traversal, then encryption will be meaningless at all.

It is assumed that the encryption is good enough, then the suffix space includes 2^N states, N is the suffix length. Considering that the attacker can only launch the attack through the Internet environment, the efficiency will be less than that of scanning the address space using Zmap [5]. We use Zmap's scanning efficiency (2^{32} addresses in 45 minutes at most) for estimation, the total traversal time for the suffix space is T hours as (4)

$$T = \frac{3 * 2^{N-32}}{4} \tag{4}$$

Therefore, to ensure security, we recommend a suffix at least 48-bit long. Traversing all addresses under the prefix will take 5.6 years, which is almost impossible in real network attacks.

In summary, the length of the prefix and suffix should be obtained by comprehensively considering the requirements of routing and verification. A /64 or /56 prefix is a general choice. In the rest of this paper, we assume a /64 prefix unless we note specifically.

D. ENCRYPTION PROCESS

The encryption described in our paper is essentially a digital signature. The message is signed by the client and verified by the server. As (1) and (2), the plaintext is a 128-bit source address, and the ciphertext is a 64-bit (or another length) destination address suffix. It should be noticed that since the mechanism is actually a signature-verification process, not a typical encryption-decryption process, we do not need our encryption process $f()$ to be reversible.

The encryption algorithm (function f mentioned in Section III.B) executed by the client is as follows:

$$H\_SA = Hash(SA) \tag{5}$$

$$P\_SA = \Phi(H\_SA, salt) \tag{6}$$

$$DA_{65-128} = e(P\_SA, key) \tag{7}$$

$$DA = strcat(prefix, DA_{65-128}) \tag{8}$$

In (5), (6), (7), (8), SA is the source address, DA is the destination address generated by the algorithm, H_SA, P_SA are the intermediate results.

The verification algorithm (function g mentioned in Section III.B) executed by the server is as follows:

$$H\_SA = Hash(SA) \tag{9}$$

$$P\_SA = e^{-1}(DA_{65-128}, key) \tag{10}$$

$$Result = \Psi(P\_SA, H\_SA, salt) \tag{11}$$

In (5) and (9), $Hash()$ is a hash function. This function is used to convert a 128-bit source address into a 64-bit sequence uniformly distributed in space. Function $Hash()$ can be arbitrary selected here, for example, md5 [26], SHA-128 [53], etc. If the sequence generated by the hash function exceeds 64 bits, we intercept 64 bits from the result. This can ensure that H_SA is evenly distributed and lacks patterns. We use md5 in our prototype.

$salt$ in (6) and (11) is the time-varying factor. Function $\Phi()$ is used to add time variability to prevent replay attacks. Here the salt is a 64-bit value that changes in different flows. Function $\Phi()$ can be arbitrary selected as well, such as $XOR()$, which is used in our prototype. And in (11), $\Psi()$ is a verification function, it is determined by $\Phi()$.

Function $e()$ is the encryption function, and $e^{-1}()$ is the decryption function. key is the encryption key of $e()$

Fig.2. shows the encryption process briefly, and Fig.3. shows the verification process briefly.

In our model, $e()$ could be symmetric encryption or asymmetric encryption. We discuss them separately.

There are many mainstream symmetric encryption algorithms, such as DES [54], AES [55], 3DES [54], etc. In our model, the ciphertext is the suffix of the destination address, which means the ciphertext should have the same length as the suffix length. As a result, DES or 3DES is very suitable for our model. (If the suffix length is not 64-bit, the encryption algorithm should be modified.) 3DES has higher security, but it costs more resources. As can be seen in the security analysis in Section IV.F, DES is already sufficiently secure, so in our prototype, we use DES as the function $e()$.

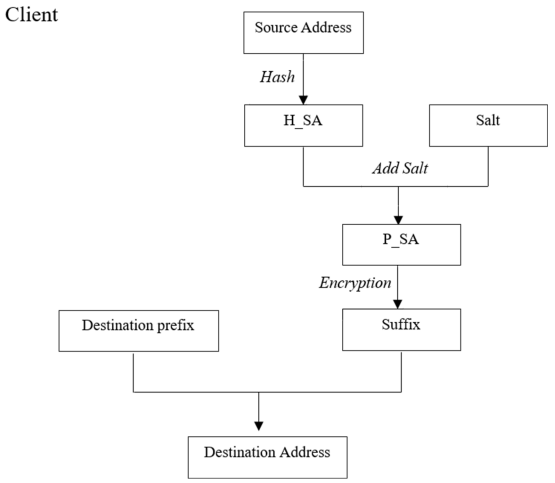


FIGURE 2. Encryption process on the client side.

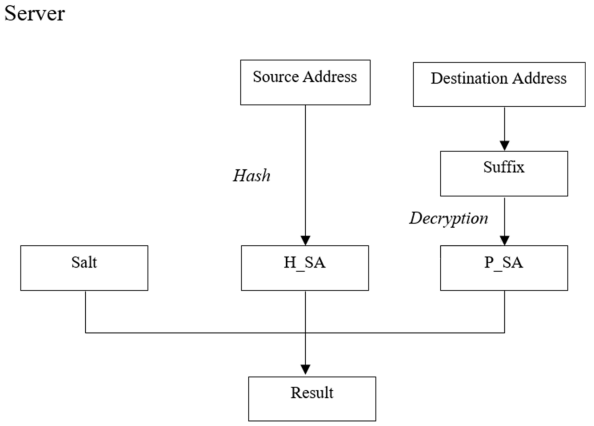


FIGURE 3. Verification process on the server side.

Besides DES, there are some very simple symmetric encryption algorithms, such as $XOR()$. In other encryption scenarios, people do not use these because they have low security levels, and can be easily cracked. In our model, this risk is related to the selection of salt. So if the generation of the salt is chosen good enough, we use $XOR()$ as the function $e()$ may not bring additional security risk. The analysis of this is introduced in Section IV.F.

Symmetric encryption can be applied in most situations. However, the encryption key and the decryption key of the symmetric encryption algorithm are the same. If the user is worried that the key transmission process may cause the risk of key leakage, or the authenticated client thinks that the IoT server is untrusted because it may leak the key, symmetric encryption is no more an appropriate choice here. In this case, we use an asymmetric encryption algorithm as function $e()$. The authenticated client saves the private key for signature, and the addressless IoT server saves the public key for verification.

Asymmetric encryption algorithms also have mature designs and implementations. RSA [56] (based on the prime

TABLE 1. Key length, Ciphertext length and security level of encryption algorithms.

Algorithm	Key length	Ciphertext length	Security level
DES	56	64	64
RSA	1024	1024	80
DSA	1024	320	80
ECC	192	192	96

factorization problem), DSA [57] (based on the discrete logarithm problem), ECC [58](based on the elliptic curve problem) can all be used here. These algorithms have advantages and disadvantages. RSA has a relatively fast verification speed. ECC can generate shorter ciphertexts with the same security level. The specific choice is determined by the demand of users.

The security level of these algorithms is described in Table 1 [59]:

The security level means the times required for brute force cracking. The security level of 64 means 2^{64} attempts are needed for cracking. In general, a security level below 64 is considered unsafe. From Table 1, it can be seen that to achieve a sufficient security level, a 64-bit ciphertext is much shorter than needed if we use asymmetric algorithms. No algorithm can guarantee the security of asymmetric encryption when the ciphertext is only 64-bit. In this case, if the public key is obtained by an attacker, it is quite easy to crack the corresponding private key, resulting in the failure of asymmetric encryption. To solve this problem, several solutions are proposed in Section III.F.

E. ADDING THE TIME-VARYING FACTOR

The generation of the time-varying factor (salt) is discussed in this subsection. This makes the replay attack no more effective, and makes the generated address no more predictable.

There are two types of salt generation methods: stateful and stateless.

1) STATEFUL FACTOR

Stateful means that the authenticated client and the server save a state-space at the same time, and hop from one state to another through a predetermined algorithm.

Here the state held by the client and the server can be arbitrary. For example, a cyclic sequence is a simple and feasible solution. The time-varying factor, which is defined as *salt* in the previous discussion, can be denoted as $s(x)$, x is a sequence of integers from 1 to N. Every time a new communication is initiated, $x = x+1$. If x is larger than N, we set $x = 1$. For example, we can set $s(x) = x$, $N = 100$, then the sequence will be 1,2,3,...,99,100,1,2,3,.... Any function $s()$ is OK, such as a codebook, or a hash function. In this way, for the same source address, the generated destination addresses have cycles of N. When N is large, it can be ensured that this cycle is difficult to be found by attackers.

Another method is to hash the previous state. The initial state is assumed as $s0$, then the first factor should be $s1 = H(s0)$. Although here $H()$ could be any function, a hash function is recommended to keep the result unpredictable. Here the n_{th} factor will be $s_n = H^n(s_0) = H(H(\ldots(H(s_0)))$. It is the result of a hash chain. This method is widely used on the Internet. For example, the temporary addresses in SLAAC Privacy Extension [60] are calculated using this algorithm.

In the stateful factor case, the client and the server need to negotiate the algorithm and the related parameters in advance. This can be done in the key exchange process.

To prevent the status sequences of the client and the server being out of sync, the factor hops only when the connection is terminated. As a result, when using this method, it should be avoided that the client uses multiple flows to visit the server at the same time. Otherwise, a more elegant hopping algorithm should be designed to avoid the sequence being out of sync.

The stateful factor case is described in Fig.4.

2) STATELESS FACTOR

A stateless factor means that the server and the client do not save any state. Only public information is used in the factor generation. The public information should be simple and verifiable, meanwhile hard to falsify. System timestamp is a perfect option that satisfies all the requirements above. So we use timestamps as our salt here.

Intuitively, the timestamp can be used as the salt directly. In this case, (6) will be:

$$P\_SA = XOR(timestamp, H\_SA) \tag{12}$$

Then (11) in the server verification process will be:

$$Result = (Time - XOR(P\_SA, H\_SA)) \in (-threshold, threshold)?True : False \tag{13}$$

In (13), the encrypted timestamp is calculated, and then compared with the local system time. If the difference is within a credible range, such as 1 or 2 seconds, then we consider the message legitimate. The attacker cannot obtain the encryption key, so even if he knows the server uses the timestamp as the "salt", he still cannot falsify the message or launch attacks.

However, using timestamp directly is not good enough. Assuming that the attacker knows the hash function in (5) (This is of large possibility because the hash function is usually public), the attacker can intercept the packet, and calculate H_SA from the source address. Assume that the granularity of the timestamp is 1 ms, the attacker can obtain 1000 possible P_SAs by enumerating all the timestamps in the past 1 second, and then forms 1000 possible plaintext-ciphertext pairs of function $e()$. One of them is true. Then the attacker can get the key by cracking. If $e()$ is good enough (such as DES/3DES), the cracking will take too long (The attacker must crack 1000 plaintext-ciphertext pairs to get 1000 possible keys, which takes at least several years if DES is used as $e()$), however, it can be better.

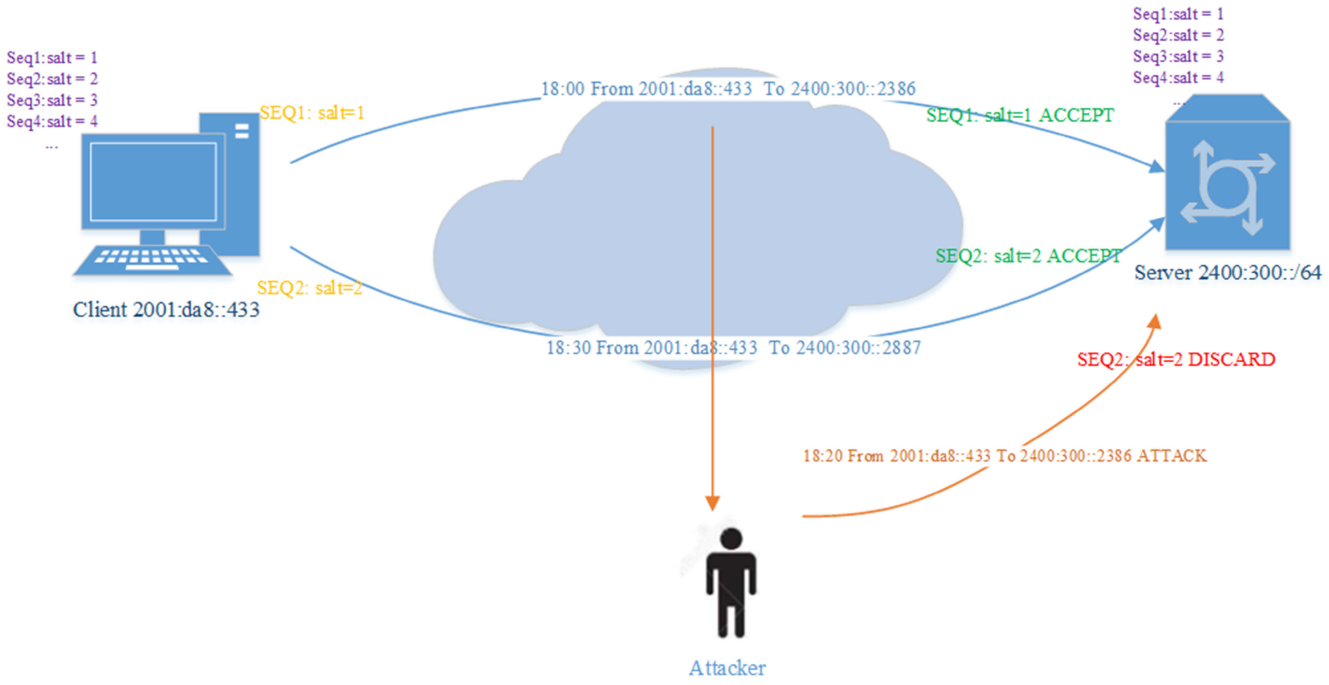


FIGURE 4. The stateful time-varying factor case. The client and the server save the same state sequence. When a connection is terminated, the salt hops to the next state. When a man-in-the-middle intercepts the packet and launches replay attacks, the destination address is already expired, making the attacks fail.

The algorithm used for generating the factor by the timestamp is as follows:

$$Salt = (SystemTime - T_0)/X \tag{14}$$

This method is similar to the one-time key generation algorithm described in RFC 6238 [61]. T_0 and X are two parameters saved by both the client and the server. T_0 is the initial value and X is the step size, which is confidential. Using this, it can be guaranteed that the salt cannot be guessed by the attacker.

The client's salting process can be described by as the following equations:

$$Salt = (TimeStamp - T_0)/X \tag{15}$$

$$P\_SA = XOR(Salt, H\_SA) \tag{16}$$

The *TimeStamp* in (15) is the system time when this function is executed. T_0 and X are the same as (14).

The server's verification process can be described as the following equations

$$P\_SA = e^{-1}(DA_{65-128})) \tag{17}$$

$$Salt = XOR(P\_SA, H\_SA) \tag{18}$$

$$T_s = Salt * X + T_0 \tag{19}$$

$$Result = (SystemTime - T_s) \in (-threshold, threshold)?True : False \tag{20}$$

Here, H_SA is the result of (9), T_0, X and *Threshold* are the same as (14).

There are several considerations of this approach.

The first is the choice of the time threshold. The threshold cannot be too small. The time difference between T_s and *SystemTime* includes transmission delay, processing delay, and system time difference between the client and the server. So the threshold should be at least higher than the transmission delay, processing delay, and the additional cost caused by network condition. At the same time, the threshold should not be too big, otherwise, the risk of being attacked will increase. In our prototype, a threshold of 5 seconds is used. We think a threshold of 2 seconds to 5 seconds is appropriate.

We should consider the system time difference between the client and the server as well. Therefore, this approach also imposes a certain requirement on the system time accuracy of the client and the server. Fortunately, this requirement is not hard to satisfy. The system time synchronization is easily done in the current Internet environment.

In the implementation of our prototype, the stateless factor approach is used. However, the stateful factor can protect the server as well, especially in some cases such as when the system time synchronization is hard to be done.

The process can be described as Fig.5. briefly.

F. ENHANCING THE SECURITY LEVEL OF ASYMMETRIC ENCRYPTION

As described in Section III.D, in order to guarantee the security level of asymmetric encryption, a longer ciphertext is needed. For example, if RSA is used as the encryption algorithm, the length of the key is better to be no shorter than

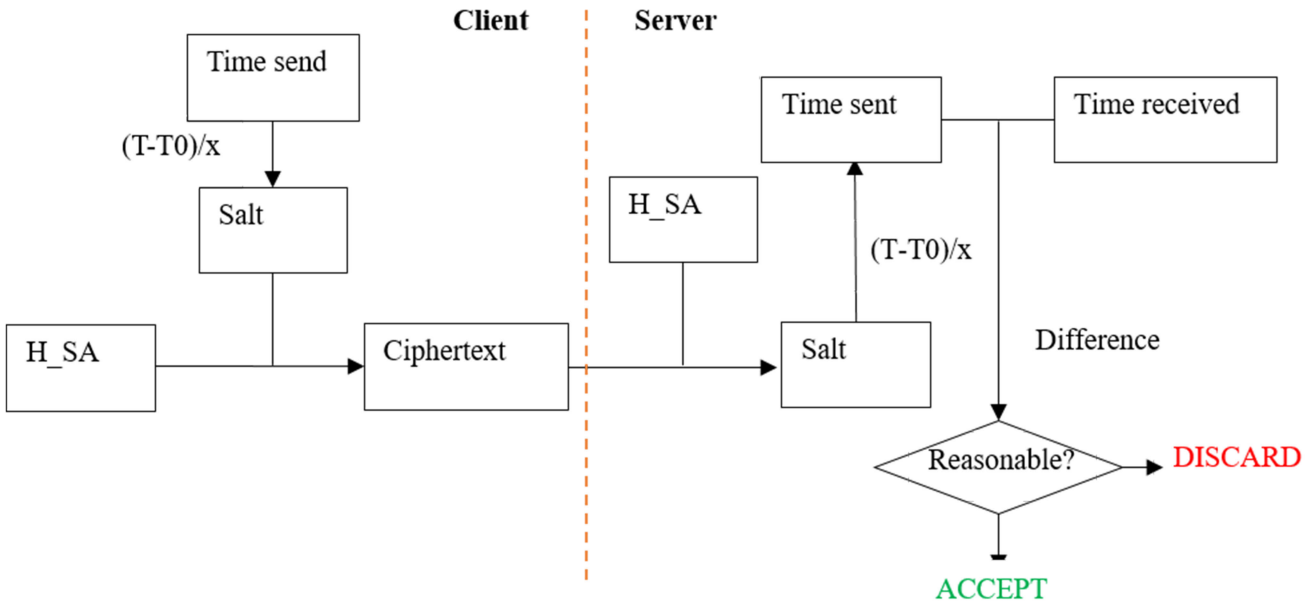


FIGURE 5. The encryption-verification process of stateless salt case. The left side is the encryption process at the client-side and the right side is the verification process at the server-side.

1024 bits, which means the ciphertext will be 1024 bits as well. Even if we use an elliptic curve algorithm, it would be better to have a ciphertext longer than 192 bits to ensure the security level. Otherwise, if the attacker obtains the public key, it will be easy to crack the private key. This is far beyond the length of the IPv6 address suffix. In this part, two approaches are proposed to solve this problem: the non-network-layer approach and the network-layer approach.

1) NON-NETWORK-LAYER APPROACH

The main idea of the non-network-layer approach is to use the payload space to provide more bits for the ciphertext. That is, part of the ciphertext which cannot be held in the suffix can be saved in the message payload. When the server receives the message, it makes the verification by the destination suffix and the additional information from the message payload.

The information is added to the payload of the first packet in order to ensure that the server does not reply to any unauthenticated message. For TCP datagrams, we use the payload of the first Syn packet, and for UDP datagrams, we add a packet before the first UDP packet to hold this information. When the server receives the message, it makes the verification. If the message is authenticated, the server accepts the following packets in the flow, otherwise, it drops the message.

Fig.6 shows the mechanism of the non-network-layer approach.

2) NETWORK-LAYER APPROACH

The main idea of the network-layer approach is to use multiple quintuples to provide more bits for ciphertext. For example, one connection can only offer 64 bits for ciphertext, then 10 connections can offer 640 bits, which is sufficient to provide enough security level for ECC or DSA algorithms.

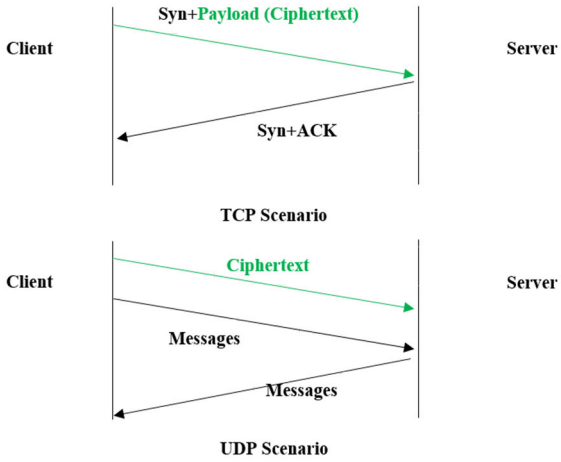


FIGURE 6. Mechanism of the non-network-layer approach. In TCP scenario, the additional ciphertext is held in the payload of the Syn packet, and in UDP scenario, the ciphertext is held in a new packet.

When a client initiates communication, it can send multiple data flows with different destination address suffixes instead of one flow. The number of flows is determined by the security requirement. Each data flow has the same source address, the same destination address prefix, and different destination address suffixes. When the server receives the data message, it combines the destination address suffixes of the multiple data flows into a single ciphertext for verification.

To combine the ciphertext correctly, the last 4 bits are used as the sequence number field. At this time, the effective ciphertext length for each flow will become N-4 bits. (N is the suffix length).

When the server receives the data message, it needs to wait for all the messages to be received before establishing the communication. For example, the client can send 1 TCP

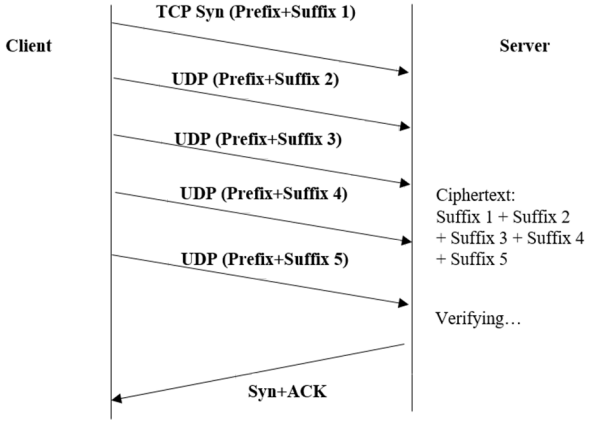


FIGURE 7. An example of the network-layer approach. The ciphertext is held in multiple destination addresses. The server combines these suffixes to form the ciphertext.

Syn packet and 9 auxiliary UDP packets, and the server receives them, reassembles the ciphertext, and conducts the verification. Then it replies Syn+ACK message to the client to establish the TCP connection.

The network-layer approach solves this problem purely in the network layer without bringing cross-layer problems. It introduces additional security benefits by establishing communication through multiple flows instead of one. However, it introduces performance problems. Communication is established only when multiple flows are all received. Packet loss in any flow may result in the failure of the communication establishment, which means that there will be more performance loss when the network condition is not desirable.

These approaches mentioned in this subsection are only used to solve the problem of insufficient ciphertext space in the asymmetric encryption scenario. Unless the client is concerned about key leakage on the server-side, a symmetric encryption solution is recommended, which can provide sufficient security at a lower cost.

Fig.7 shows an example of the network-layer approach.

G. MATCHING THE KEY OF DIFFERENT USERS

When an IoT server serves multiple authenticated users, the server will hold multiple keys. In this subsection, we discuss how to match the key used by the specific user when receiving a data message.

The most intuitive method is to traverse the keys. The server selects one key to make the verification. If the verification fails, the server makes another attempt using the next key until the verification succeeds or all the keys are tried. If the number of users is small and the encryption algorithm is not complicated, the additional time cost for verification is not large.

To improve the efficiency of the above method, the Most Recently Used algorithm can be used. Using this, the server makes the verification with the most recently used key first. There are many specific implementations of this algorithm, for example, the server maintains a queue of keys, and each time a key is used, it is relocated to the first position in the queue. When a data packet arrives, it is verified sequentially according to the order in the queue. This algorithm is similar to the LRU algorithm [62] which is used to improve the efficiency of memory replacement in the operating system. This depends on an assumption: some users access the server more frequently, and some users rarely access the server. This is usually true. In this case, changing the order of the keys for verification can increase the verification efficiency and reduce the matching time.

There is a problem with matching keys one by one. If the server is under a DDoS attack, such as Syn-Flood, each data packet needs to match all keys before it is discarded. This may weaken our advantages in preventing DDoS attacks to some degree. So this method is suitable only when the number of clients is small. However, for an IoT server, generally the number of users is not big.

Another idea is to design an independent ID to each user device when generating and distributing the encryption keys. When the client sends the data message, the ID is also embedded in the address information and sent to the server. We introduce two ways using this idea:

1) The port space can be used for user identification. For security and functionality reasons, we avoid the widely used ports and the low ports. The port space of TCP and UDP is 16 bits. Except for the low ports and the widely used ports, there are still more than 50,000 ports available. This is enough for most IoT devices.
2) Part of the suffix can be used to hold the user's ID. For most IoT servers, there are not too many users, so a small part of the address space, such as 8-bit is sufficient. This can provide space for 256 authenticated clients, and the rest space is enough for encryption.

If the second method is used, the destination address should be

$$DA = stract(prefix, R(ID + e(P\_SA, key)) \tag{21}$$

$R()$ is a reversible function (it is better credential to prevent the attacker from using metadata analysis to get the patterns of the user id). The server uses R^{-1} to get the suffix, extract the user ID, and check whether there is a key corresponding to the user ID. If the server finds the key, then it decrypts the message using the corresponding key. Otherwise, the message is discarded by the server.

The above algorithms can also be combined to provide services to a large number of users. In this case, each client is assigned a non-unique user ID. When the client initiates connections, it uses part of the destination address suffix space and/or the port space to transmit the user ID, and if there are multiple keys corresponding to one user ID, then the server matches the key one by one or using a Most Recently Used algorithm. This can help the server to serve a large number of clients.

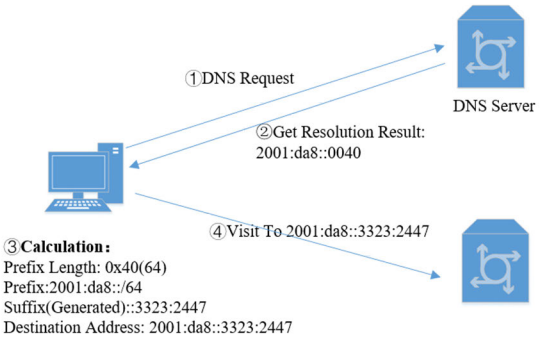


FIGURE 8. Mechanism with DNS. The client sends the DNS request and gets the resolution result first, and then calculates the destination address.

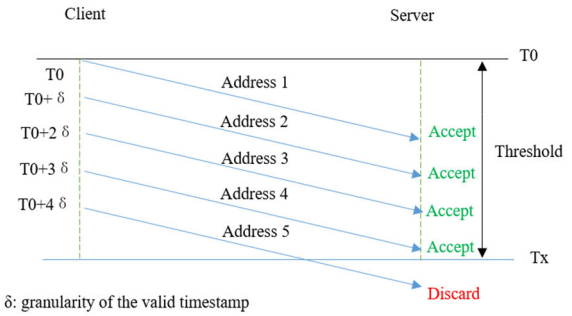


FIGURE 9. There are more than one legitimate destination addresses in the stateless salt case. Address 1,2,3 and 4 are all considered legitimate.

H. DOMAIN NAME SYSTEM

Sometimes users do not want the client to access the IoT server directly through the IP address. In this case, we introduce the domain name system to the addressless server.

Our design is fully compatible with the existing DNS system. Unlike traditional servers, our server uses an IPv6 prefix instead of an IPv6 address. The domain name is pointed to an address under the prefix, for example, an all-zeros suffix address, such as 2001:da8:c597:2426::. This address does not provide services. When an authenticated client initiates communication to the server, it gets this address using DNS and extracts the prefix. Then it generates the suffix using the algorithm described previously. When the network condition of the server changes, it can modify the address. Our model can be more flexible using this method.

DNS can also be used when it is necessary to change the prefix length in some scenarios. The end 8-bit of the address is used to hold the value of the prefix length. Because the length of the prefix is smaller than 128, it takes 7 bits at most. This will not affect the prefix. After the client gets the DNS response, it extracts the prefix and the prefix length, then generates the packets using this information.

It should be noticed that the client cannot get the prefix length using the address with an all-zero suffix. An address like 2001:da8:c597:2426:0000:0000:0000:0000 may have a prefix as 2001:da8:c597:2426::/64, or a prefix as 2001:da8:c597:2426::/65. The prefix length cannot be determined by the number of zeros.

With DNS, the server can provide shorter or longer prefixes as needed, making the mechanism more flexible.

Fig.8. shows the addressless server access process using DNS.

IV. SECURITY ANALYSIS

In this section, the security features of the addressless server model are analyzed.

A. NETWORK SCANNING

Address scanning is always an effective way to locate the server and launch attacks. Because of the difference between the address space of IPv4 and IPv6, it is impossible to scan the whole IPv6 address space although it takes only about 45 minutes to scan the entire IPv4 address space. However, as described in Section II, there are already some approaches in IPv6 scanning.

IPv6 scanning from the Internet can be simply divided into two types:

1) Scan by the collected IPv6 address records
2) Generate a hitlist using machine learning algorithms.

In the first approach, the scanner first collects the IPv6 addresses from the Internet or LAN records, such as web access records, traffic monitoring records, routing information records, DNS information, and DNSSEC information, etc. In the second approach, the scanner uses some real IPv6 addresses as the seeds, generating hitlists of target IPv6 addresses by pattern recognition algorithms.

As described in Section III, the brute force scans cannot pose any threat to addressless server. The time it takes to scan the entire IPv6 address space under an IPv6 prefix (unless the prefix is too long) by brute force is too long to harm our model. However, in section III.E, an approach is proposed to add a stateless time-varying factor in our model. In this approach, there will be a legitimate time window, all timestamps in the time window will be considered legal. As a result, there will be more than one legal address among the 2^N addresses (Here N is the suffix length). It is assumed that there are P legal addresses within the threshold. (For example, if the threshold in (20) is 1.5 seconds, and the step size in (15) is 1.5 ms, then there will be 1000 destination addresses that can pass the verification.) In this case, the hit probability of a single scan will be $\frac{P}{2^N}$.

Fig.9. shows that in the stateless salt case, there are more than one legal destination addresses corresponding to the same source address. In Fig.9, address 1 to address 4 are all legal, because there is more than one legal salt in the time window.

Furthermore, if there are Q authenticated users' keys in the server at the same time, and the matching algorithm described in Section III.G is conducting the verification one by one or hiding the user ID in the destination address suffix, the hit probability of a single scan will be (Considering that

the legal address is changing all the time, each scan can be regarded independent)

$$Pro = \frac{P * Q}{2^N} \tag{22}$$

The expected time T of the scanning will be (assume the efficiency of the scan is the same with Zmap in IPv4) (in hours)

$$T = \frac{3 * 2^{N-32}}{4 * P * Q} \tag{23}$$

To prevent the server from being scanned, T should be long enough, for example, it can be more than 1 year. As a result, P, Q, N should satisfy the following equation: (Note that if the client and the server use ports to transmit user ID, then Q should not include this part of data.)

$$N - log_2 PQ \geq 46 \tag{24}$$

Scanning by the collected IPv6 addresses is obviously ineffective to the addressless server. A message will be replied only when the source address and the destination address pass the verification. Otherwise, no packets will be responded to. Obviously, the collected IPv6 addresses cannot pass the verification, making this kind of scanning ineffective.

Scanning by generating a hitlist using a machine learning algorithm is also ineffective to the addressless IoT server. The essence of this method is to learn the patterns existing in the addresses by data analysis. In Section IV.G, the big data characteristics of our generated addresses are analyzed. It can be seen that the addresses generated by our algorithm are random enough. The attacker cannot use this approach to launch attacks.

In addition, another type of scan is initiated from the subnet. The attacker collects the addresses through the ND information, then launches scans or attacks. This is ineffective to the addressless IoT server as well. Other devices in the subnet cannot obtain an IPv6 address of the server except the link-local address, however, the link-local address does not respond to any application, and attacks cannot be launched through it.

In summary, for an addressless IoT server, it cannot be detected by any scanning method. The outside devices cannot discover the server through any active scanning, let alone launching an attack by scanning.

B. DoS ATTACK

DoS attacks include many types. The most popular types are TCP Syn-Flood, UDP Flood, etc. Some attacks have no damage in the IPv6 network environment, and some have nothing to do with our model, such as smurf attacks. We do not discuss them in this subsection.

Syn-Flood is no longer harmful to addressless servers. In our model, all syn packets received will be verified according to the source and destination address, and the unauthenticated ones will be dropped immediately. In this case, no Syn+ACK message will be sent, and the server will not allocate any storage or CPU time to wait for the subsequent ACK messages. Therefore, Syn-Flood will not cause any harm to the server.

UDP Flood does not cause any harm for the same reason. Any unauthenticated UDP packets will be dropped immediately, the application will not allocate any resources to wait for subsequent data packets. As a result, UDP Flood will not cause the server to be overloaded.

However, restricted by hardware conditions, our mitigation of DoS attacks is also limited. Our model is powerless against attacks such as bandwidth exhaustion.

C. APPLICATION VULNERABILITY ATTACK

Many network attacks are aimed at application-layer vulnerabilities. For example, in a SQL injection attack, an attacker injects malicious code into a SQL request to launch the attack.

Our model can prevent this from happening. Through the encryption mechanism, the server only responds to authenticated clients. The attacks aimed at application vulnerabilities are prevented at the network-layer, the corresponding attack packets will not be sent to the application or operating system, so they will not cause security problems.

However, this is based on the premise that the authenticated users will not launch malicious attacks. Our model cannot prevent an authenticated user from launching such attacks. Therefore, the service provider should prevent malicious clients from obtaining authorization. Besides, it is also possible to hijack the client and launch attacks on the server. To prevent this from happening, the client needs more security protection methods, such as the addressless client model which will be discussed in another paper of ours.

D. REPLAY ATTACK

As described in Section III.E, the time-varying factor is added to the algorithm to make the addresses generated each time different from other addresses. Using this, the server can be prevented from replay attacks.

However, if a man-in-the-middle intercepts a packet, he can launch a replay attack within a time threshold. To prevent this from happening, if the server receives a new flow with the same address that has been used within a time threshold, it can regard this packet unauthenticated and drop it, although this kind of attack can hardly be successfully launched because the time window is too small. However, this method requires that the client would not initiate multiple flows within a time granularity described as X in (15) and (19) in Section III.E, so that the salt in different flows can be different.

E. CONNECTION MONITORING AND SESSION HIJACK

Our model cannot prevent malicious middleman from obtaining the quintuples that are being used by monitoring the connection. Although the man-in-the-middle cannot initiate a scan or a replay attack through this quintuple, he can attack or hijack the connection. For example, a man-in-the-middle may forge a malicious packet with the same quintuple

to launch a session hijack or an attack on the connection such as reset the connection using TCP RESET packet.

Our model cannot prevent this directly. Since our verification is performed on a flow, when a connection has been established, we will no longer conduct the verification on the following packets. However, encryption at the application layer, such as TLS [63], DTLS [14], HTTPs [52], etc. can encrypt the payload of the packets and prevent the connection attacks and the session hijacks. It can protect user privacy as well. These methods are performed at the application layer, which is fully compatible with our model which is performed at the network layer.

In addition, in Section III.B, encryption at the packet level is discussed. This causes each packet to have a different destination address, and the verification is performed to every packet. This can prevent the attack on connections and session hijack as well. However, these attacks can be easily prevented by using TLS/DTLS without introducing other problems brought by the verification on each packet, so we think there is no need to use packet-level encryption and verification.

As a result, our model can prevent the server from being monitored and launched connection attacks by combing with application-level encryption such as TLS [63].

F. KEY CRACKING

The security of symmetric encryption depends on the algorithm. Usually, the attacker can intercept the plaintext-ciphertext pairs to guess the key. In our model, it is meaningless to intercept a message, because we add a time-variant salt in the encryption process, which ensures that the man-in-the-middle cannot obtain the plaintext of the function $e()$ from the source address directly. The man-in-the-middle can only obtain the source address and the destination address, and it is impossible to infer the key from this information.

Obtaining multiple plaintext-ciphertext pairs will make cracking easier. However, in our scenario, this is obviously impossible. Different flows will carry different destination addresses even if these flows are sent by the same device. This prevents the middleman from testing the key obviously. Therefore, although many people think that DES encryption is not safe enough in the current Internet, however, in our scenario, DES is enough to protect the security of the IoT server.

In extreme cases, even if we use the simplest symmetric encryption algorithm, such as $XOR()$, it is hard for the attacker to crack the key. The plaintext we use in encryption is hashed and salted, the salt we use is confidential (in the stateless case, the parameter T0 and x is confidential), and the ciphertext distribution is random enough (we will discuss it in Section IV.G), so even we use the simplest encryption algorithm, it is hard for the attacker to crack the key. This will reduce the load of the device, but from the perspective of multiple protection, we prefer to use popular encryption methods such as DES, unless the computing resource is limited.

In some cases, we use asymmetric algorithm instead of the symmetric algorithm in the encryption. This may due to the concern that the server can leak keys, or there is no need to keep the server-side key secret (such as using the PKI system [64] to distribute user public keys to provide large-scale services, which will be discussed in future papers). For asymmetric encryption algorithms, the cracking method is to calculate the private key by the public key. To prevent this, a long enough private key is required, resulting in the private key length and the ciphertext length much higher than symmetric encryption, and the time cost in encryption and decryption much bigger. How to ensure the security level of asymmetric encryption is discussed in Section III.F.

G. TRAFFIC METADATA ANALYSIS

Attackers can obtain the pattern of addresses by big data analysis on the traffic. This can reduce the scanning space and increase the possibility of attacking or key cracking. This is a threat to server security. To fight against it, we make the suffixes generated by our algorithm more unpredictable and evenly distributed. With the help of this method, the traffic data analysis can be less harmful.

Simulations are made to generate the suffixes by our algorithm to demonstrate it. In the simulation, we generate multiple addresses to analyze the big data characteristics. The simulations are made in three aspects, and the results are shown in Fig.10.

1) Group A: In Group A, 1000 addresses are generated at different time using the same source address. The distribution of the suffixes is shown in Fig.10.(a). This demonstrates that the addresses generated by the same device at different times are sufficiently random.
2) Group B: In Group B, 1000 different active addresses under the same /64 prefix are collected in the campus network of Tsinghua University as the source addresses, and we generate 1000 destination suffixes using these source addresses. The result is shown in Fig.10.(b). The red points stand for the input addresses, while the blue points stand for the generated suffixes. It can be seen from the result that the 64-96 bits of the source addresses are all around 0, and the 97-128 bits are concentrated as well, while the generated suffixes have good random characteristics. The result demonstrates that the addresses generated by different devices in the same subnet are sufficiently random.
3) Group C: In Group C, 100k real unique IPv6 network addresses in the campus network of Tsinghua University are collected as source addresses, and 100k destination suffixes are generated using these. Fig.10.(c) shows 1000 source address-destination address pairs. The red points are the collected source addresses while the blue ones are the generated suffixes. The result demonstrates that the collected IPv6 addresses have more obvious patterns than the generated ones. The generated suffixes are random enough.

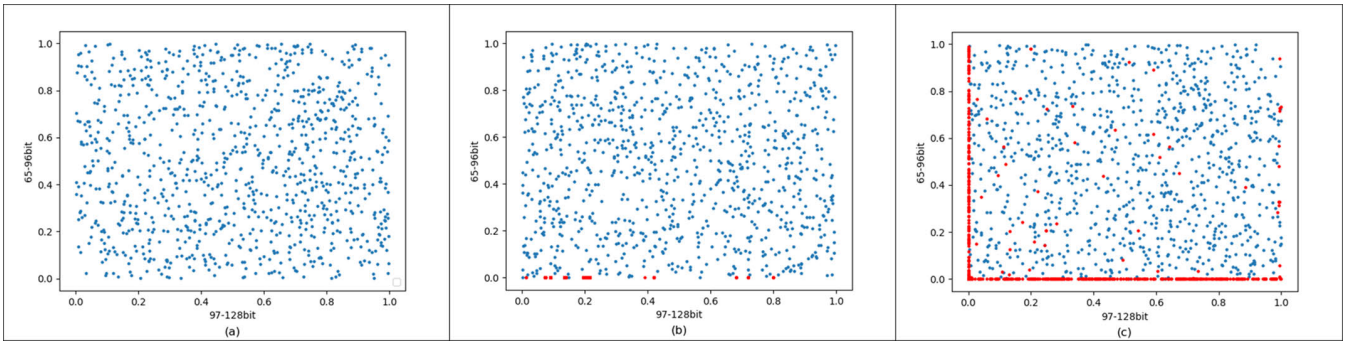


FIGURE 10. The distribution of the generated suffixes. (a) is the result of Group A, (b) is the result of Group B, (c) is the result of Group C. In (b) and (c), the red points are the source addresses and the blue points are the generated addresses. It can be seen that the addresses generated by our algorithm are random and evenly distributed.

TABLE 2. KS-test result of generated suffixes.

	Generated Suffixes of Group A	Generated Suffixes of Group B	The Suffixes of Source Addresses of Group B	Generated Suffixes of Group C	The Suffixes of Source Addresses of Group C
D value	0.0032	0.0052	0.9806	0.0027	0.8106
p value	0.7696	0.6763	0.0000	0.8678	0.0000

We can also use entropy to describe the randomness of addresses more exactly. The entropy here is calculated for each nybble[3] to evaluate the randomness of the address [38]–[41]. That is, if character c_i occurs N_i times in the k-th nybble (assume there are N samples totally), the entropy of the k-th nybble is

$$e_k = -\frac{1}{4}\sum_{i=1}^{16}\frac{N_i}{N}log(\frac{N_i}{N}) \tag{25}$$

Fig.11 shows the results. Similar to Fig.10, Fig.11 (a) is the entropy of 1000 addresses generated by the same device at different times. Fig.11 (b) shows the entropy of 1000 different addresses generated by different devices in the same subnet. The red line stands for the entropy of the 1000 source addresses' suffixes, while the blue line stands for the entropy of the generated suffixes. Fig.11 (c) shows the entropy of 100k suffixes generated by 100k different devices. The red line stands for the addresses in real traffic while the blue line stands for the generated suffixes. It can be seen that regardless of the randomness of the source addresses, the nybbles in the suffixes generated by our algorithm are random enough.

Furthermore, KS-test is used to test whether the generated addresses follow a uniform distribution. The result is shown as table 2.

From Table 2, it can be seen that the p-value of the generated suffixes in Group A, B, and C are greater than 0.05, while the p-value of the suffixes of the source addresses are less than 0.05. This demonstrates that our generated suffixes follow the uniform distribution, while the suffixes of the source address do not.

The results above show that the addresses generated by our algorithm are random enough and evenly distributed, which means the addresses are unpredictable. As a result, the traffic data analysis can be less harmful.

H. ND RELATED ATTACK

At the access layer, ND (Neighbor Discovery) [3] attacks should be considered as well. Although the ND protocol solves the ARP-related security problems in IPv4, it introduces new security problems in IPv6, such as RA spoofing, malicious redirection, cache overflow, and so on. We can divide the ND-related attacks into three types: spoofing attacks (RA spoofing, malicious redirection, etc.), cache exhaustion attack, and DAD attack [65], [66].

Our model can defend against part of spoofing attacks. Some ND packets have no effect on the server after the prefix is assigned and the route is configured. In this case, this kind of spoofing attacks cannot harm the server. However, if our model uses DHCP-PD to delegate the prefix, there may still be configuration errors caused by RA spoofing attacks in the delegation process, making the server not being able to communicate with the outside devices correctly. Therefore, our model cannot prevent all types of spoofing attacks.

The cache exhaustion attacks will be less harmful to our model. In our model, a server is allocated a prefix instead of an address. So false NS and NA messages will be considered invalid because they claim that the server has an address, not a prefix. Therefore, traditional cache exhaustion will not harm the addressless IoT server. However, there may be new types of cache exhaustion. For example, an attacker can maliciously apply for /64 prefixes to exhaust the routing table cache or prefix resources.

DAD DoS attacks [67] cannot pose a threat to our model. DAD is used to prevent two devices from being configured with the same IPv6 address. However, in our model, there is only one device under a prefix, DAD messages are no longer needed. Therefore, the DAD DoS attack will no longer be harmful.

[3]A nybble is 4-bit. It is a character in hex.

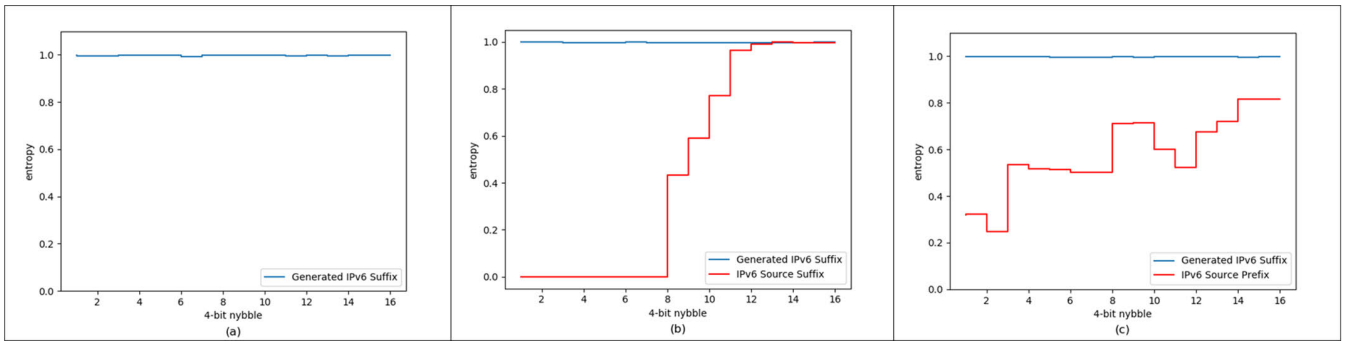


FIGURE 11. The entropy of the generated suffixes. (a) is the result of Group A, (b) is the result of Group B, (c) is the result of Group C. In (b) and (c), the red line is the entropy of source addresses and the blue line is the entropy of generated addresses. It can be seen that the addresses generated by our algorithm are sufficiently random.

However, ND attacks to the link-local address may still be effective because link-local addresses are used in our model. In this case, although our model can alleviate the harm of the ND attacks on non-link-local addresses, it is still a better choice to deploy ND-related security policies in the subnet, such as SEND [68], [69], RA-guard [70], [71], etc.

V. OTHER DESIGN CONSIDERATIONS AND DISCUSSIONS

In this section, other features of our model are discussed.

A. TRANSPARENT APPLICATION SUPPORT

There is no application-layer modification in our model. All changes are made at the network-layer, so theoretically, most applications can be transparently supported.

However, there may be some applications that require the client to initiate multiple connections to the server at the same time. In our model, the connections may have different destination addresses. In most cases, this will not affect the applications. It is difficult to imagine that there are applications whose background logic depends on whether the destination addresses in different flows are consistent. However, there are diverse applications running on the Internet, it is difficult to ensure that all applications meet this requirement. As a result, our model may not support the applications whose logic depends on the consistency of the destination addresses in different flows.

Besides, although it is usually not necessary to respond to ICMP messages in the non-subnet scope for IoT servers, sometimes users also need the server to reply to ICMP ping messages. There are two solutions to allow the server to reply ICMP ping packets:

One is that the server only replies to the ICMP messages sent by authenticated clients. The other ICMP ping packets will not be replied. In this case, only the authenticated clients can find the server, which guarantees the server's imperceptible feature. This can be done by the encryption algorithm described in Section III. In our prototype, this approach is implemented.

The other solution is to select an address under the prefix, and use it as an ICMP beacon. This address only replies to ICMP packets, and discards all other messages. It brings management benefits. However, this may destroy the server's imperceptible feature.

B. COMPARISON WITH OTHER IoT SECURITY SOLUTIONS

In this subsection, the differences of our model compared with other related work about IoT security are discussed.

1) Compared with other solutions, our model has different application scenarios. Other security solutions are focused on specific scenarios, for example, 6LowPsec [9] is based on 6LowPAN [8], which is a solution of wireless personal area networks; SSATP [16] is a solution in the communication between the Edge nodes and Fog nodes in Fog computing. In our model, encryption is conducted inside the IPv6 address itself. It can be used in any Internet scenario that allows IPv6 Prefix Delegation. Considering that the current DHCPv6 supports Prefix Delegation Model, supporting PD does not cause difficulties in deployment. As a result, our model can be widely used in industrial IoT, home IoT, and other scenarios.
2) Our model works on a different layer with the other solutions. Our model is a network layer solution and can work without modifying the access network environment. It is compatible with other security models that support IPv6. Other models usually work on other layers. For instance, 6LowPsec [9] works on the data link layer and the sublayer. TLS, DTLS and related protocols such as SSATP [16] and LSSP [17] work at transport layer and application layer. As a result, our model is independent of these solutions and compatible with the protocols at the other layer, such as SSATP or LSSP.
3) Our model is also different from other secure protocols at the network layer, such as IPsec [6]. IPsec encrypts the whole payload while our algorithm encrypts the address, so our model is also compatible with IPsec. Encrypting only the address is more lightweight than encrypting the entire payload. Furthermore, in some scenarios, it is required to check the payload. In this

case, only our model can be used because the payload is not encrypted.

4) The other solutions, for example, 6LowPsec [9], etc. are usually based on some IoT architectures. They propose lighter secure protocols than traditional Internet protocols. Our work is directly based on IPv6, it is still a novel model on the modern Internet. It does not overlap with traditional security solutions.

Our model can comprehensively deal with most of network security issues. Because our model only introduces additional operations during connection establishment, and the encrypted content is only carried in the IP address suffix, it will introduce very small overhead. It is a model suitable for most IoT scenarios.

C. COMPATIBILITY, SIMPLICITY, AND EVOLVABILITY

Compatibility with existing networks is very important. For a new model working on the Internet, compatibility with existing Internet must be guaranteed, otherwise large-scale deployment will face great difficulties. As mentioned above, our model can be used smoothly in the existing Internet without any modification on the network environment. The clients and servers can be used anywhere on the Internet without the cooperation of routers, ISPs, DNS service providers, etc. The only requirement is to allocate a prefix to the server. As a result, our model has a very low deployment cost.

A good mechanism should be simple enough and easy to use, otherwise it will bring obstacles to large-scale deployment in the complicated modern Internet environment. At the same time, a sufficiently simple mechanism also has good stability. A complex mechanism is more likely to be limited by boundary conditions. It can be seen that the idea of addressless IoT server is very simple. The model has a clear idea, and it is easy to understand. This simplicity ensures that the mechanism is easy to apply, and will not cause obstacles in deployment.

Our model eliminates the one-to-one correspondence between server and IP address, and uses the IPv6's huge address space and encryption algorithms to ensure that only authenticated clients can visit the server. Similar encryption has never been performed before. We apply it at the network-layer to better complement the existing network security model. This provides a rich evolution space for subsequent expansion. Researchers can propose more algorithms based on this model.

D. IPv6 ADDRESS SPACE CONSIDERATION

IPv6 is designed to provide more address space to alleviate the insufficiency of IPv4 address space. However, in our model, we assign a prefix to each IoT server. Will the IPv6 addresses be insufficient if a prefix is allocated instead of an address to a server? This worry is unnecessary. There are 8.6 billion /36 global unicast prefixes, which means that everyone in the world can be allocated a /36 prefix. What's more, there are 268 million /64 prefixes under a /36 prefix, which means that the current IPv6 address space is sufficient to allocate 268 million /64 global unicast prefixes for everyone. Even if a /56 prefix is allocated to each server, the current IPv6 address space can offer everyone 1 million /56 prefixes. Therefore, we think IPv6 addresses in the world are sufficient in our model.

E. IPv4 CONSIDERATION

Although our mechanism is highly dependent on the large address space provided by IPv6, our mechanism itself is not IPv6-specific. Theoretically, our model can also work in IPv4. An IPv4 server can be assigned a segment of addresses and listen on all these addresses. Authenticated clients can generate the destination address suffix using the algorithm described in Section III.

However, this does not make sense. There are two reasons:

1) IPv4 address space is too small, and the IPv4 addresses are scarce. It is too wasteful to assign a segment of addresses to an IPv4 server.
2) The security level of the encryption cannot be guaranteed with a too-short ciphertext. The ciphertext space can be easily traversed, resulting in the model not being able to provide enough security protection.

Therefore, our model is only suitable for IPv6.

F. CONSIDERATIONS ABOUT NON-IoT SERVER

As our previous discussion, our model can be used not only on the IoT servers. Any private server that provides non-public services can use our model to enhance security. However, IoT servers usually provide services to a limited number of clients, and they are usually deployed in different network environments. At the same time, IoT servers are usually not well protected like the public servers. As a result, our model is very suitable for IoT servers. Of course, some non-IoT servers, such as private cloud servers, can also use our model to protect security.

G. LIMITATION OF ADDRESSLESS IoT SERVER

The Addressless IoT Server model has the following limitations:

1) Our model is designed based on IPv6, so our model may not be compatible with some data link layer or physical layer protocols in which the IPv6 addresses are compressed. Neither is our model compatible with protocols that do not support IPv6.
2) Our solution trusts authenticated devices. If an authenticated device launches a malicious attack, such as DoS attack, fragment attack, or SQL injection, it may harm the device. Therefore, it is necessary to ensure that the authentication is carefully conducted.
3) Our solution believes that the key exchange process is credible. However, this may be a challenge.

VI. IMPLEMENTATION AND EXPERIMENT

In this Section, our implementation of addressless IoT server and the experiments based on our prototype are introduced.

A. PROTOTYPE IMPLEMENTATION

Our prototype is implemented based on Linux. DHCP-PD is used for prefix delegation, and the NetFilter Linux kernel module is modified to implement our mechanism. In our prototype, the following features are implemented:

1) The server is assigned a prefix and listen on all the addresses under it.
2) The authenticated client and the server save the key pair. The client uses the algorithm described in this paper to generate a destination address in communication, and the server performs verification by the addresses.
3) Both symmetric encryption (DES) and asymmetric encryption (RSA with a 1024-bit key, ECC with a 192-bit key) in the algorithm.
4) To protect the server from replay attacks, a stateless factor is used as the salt.

The algorithm implemented in the prototype is described as follows. In the case of symmetric encryption, the algorithm on the client-side is Algorithm 1, while the algorithm on the server-side is Algorithm 2:

Algorithm 1 Client-Side Algorithm in Prototype (Symmetric Encryption Case)

Input: SourceAddress,Key,Prefix,T0,X
Output: DestinationAddress

```
H_SA = md5(SourceAddress)[0:64]
T_current = time.currenttime()
Salt = (T_current − T0) / X
P_SA = XOR(Salt, H_SA)
Suffix = DES(P_SA,Key)
DestinationAddress = strcat(Prefix, Suffix)
return DestinationAddress
```

In the case of asymmetric encryption, the algorithm on the client-side is Algorithm 3, while the algorithm on the server-side is Algorithm 4:

In the algorithms described above, *SourceAddress* is the address of the client. is the source address of the communication quintuple, while *DestinationAddress* is the destination address of the quintuple. *Key* is the encryption key, T_0, X, and *Threshold* are parameters to generate the salt.

If other encryption algorithms are used here, it is only necessary to replace the *DES*() in Algorithm 1,2 or *RSA*() in Algorithms 3,4 with other encryption algorithms, such as ECC.

B. EXPERIMENT ENVIRONMENT

Our experiment is built on a /48 subnet under 2001:da8::/32, China Education and Research Network. DHCP-PD is used

Algorithm 2 Server-Side Algorithm in Prototype (Symmetric Encryption Case)

Input: SourceAddress,DestinationAddress,Key,T0,X, T_threshold
Output: True/False

```
H_SA = md5(SourceAddress)[0:64]
Suffix = DestinationAddress[64:128]
for key in keys(): do
    P_SA = DES^-1(Suffix, Key)
    T_current = time.currenttime()
    Salt = XOR(H_SA, P_SA)
    T_send = Salt*X + T0
    T_delta = T_current - T_send
    if T_delta < T_threshold and T_delta > −1*T_threshold then
        return True
    end if
end for
return False
```

Algorithm 3 Client-Side Algorithm in Prototype (Asymmetric Encryption Case)

Input: SourceAddress,PrivateKey,Prefix,T0,X
Output: DestinationAddress,Payload

```
H_SA = md5(SourceAddress)[0:64]
T_current = time.currenttime()
Salt = (T_current − T0) / X
P_SA = XOR(Salt, H_SA)
CipherText = RSA(P_SA,PrivateKey)
Suffix = CipherText[0:64]
DestinationAddress = strcat(Prefix, Suffix)
Payload = CipherText[64:]
return DestinationAddress
```

Algorithm 4 Server-Side Algorithm in Prototype (Asymmetric Encryption Case)

Input: SourceAddress,DestinationAddress,Payload, PublicKey,T0,X,T_threshold
Output: True/False

```
H_SA = md5(SourceAddress)[0:64]
Suffix = DestinationAddress[64:128]
CipherText = strcat(Suffix,Payload)
for key in keys(): do
    P_SA = RSA^-1(Suffix, PublicKey)
    T_current = time.currenttime()
    Salt = XOR(H_SA, P_SA)
    T_send = Salt*X + T0
    T_delta = T_current - T_send
    if T_delta < T_threshold and T_delta > −1*T_threshold then
        return True
    end if
end for
return False
```

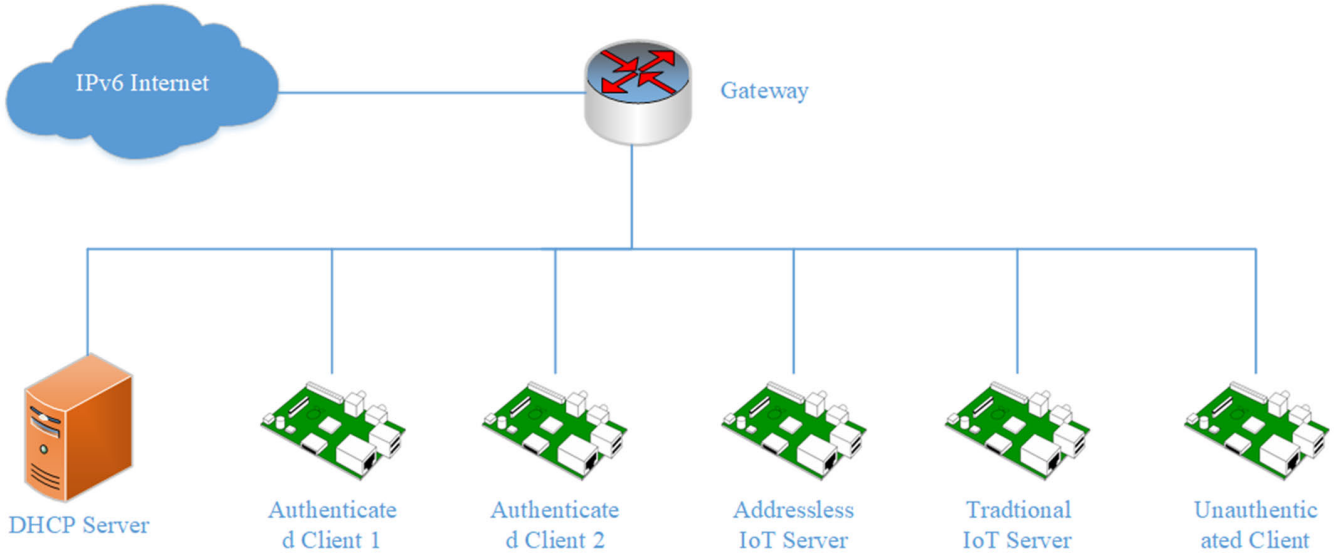


FIGURE 12. The experiment network topology, in which two Pis are configured as authenticated clients, one is configured as unauthenticated client, one is configured as Addressless server, and the other is configured as traditional server as control group.

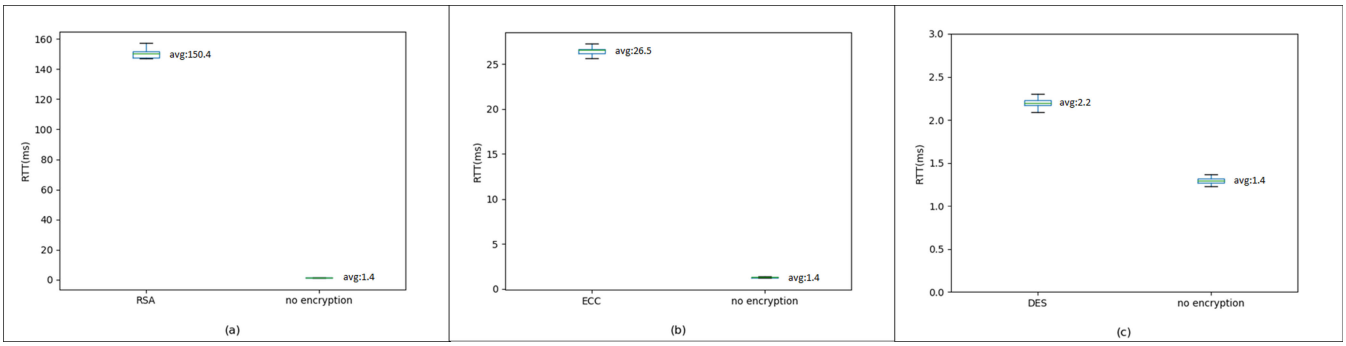


FIGURE 13. Results of the RTT experiment on the first packet in the flow. (a) is the result of RSA, (b) is the result of ECC, and (c) is the result of DES. The figure is plotted in boxplot.

for prefix delegation in our experiment subnet. Kea is used for the DHCP server, which is configured in DHCP-PD mode.

Raspberry Pis are used as experiment devices. Our experiment is based on Linux, specifically, on the Raspbian OS that comes with the Raspberry Pi. This OS is based on Debian.

Experiment topology is briefly described as Fig.12.

Two authenticated clients and one unauthenticated client are deployed in the experiment subnet. An IPv6 addressless server is configured with a /64 prefix in the same subnet. A traditional server is also deployed in the subnet as the control group.

C. EXPERIMENT ON SECURITY

Experiments on security are firstly conducted. ICMP ping, SSH (based on TCP port 22), and HTTP (based on TCP port 80) are used as the application-layer protocols to access the server. The results show that the authenticated client can visit the server smoothly, while the unauthenticated client cannot access the server, and it cannot get any response either. This shows that the server cannot be perceived by unauthorized devices.

To test the security of the server, an unauthenticated client is used as an attacker. It uses scanning tools to scan the server. A 100-hour brute-force scan to the server is performed. The result shows that the scan does not hit any address.

Then, 1 million addresses under the /64 prefix are generated using Entropy/IP [39] and 6gen [38]. They are used as the hitlists to scan the server. The scan does not hit any address as well.

We collect ND information from other devices in the subnet, and conduct scans based on the ND information. The result shows that we cannot obtain the addresses which can be accessed. Attacks cannot be initialed in this way. Therefore, our model can prevent the server from being scanned by existing IPv6 scanning approaches.

D. EXPERIMENT ON PERFORMANCE

In this subsection, our experiments on performance are introduced. Experiments on RTT are conducted firstly because the encryption process affects the delay most. Besides RTT, experiments on bandwidth are conducted as well.

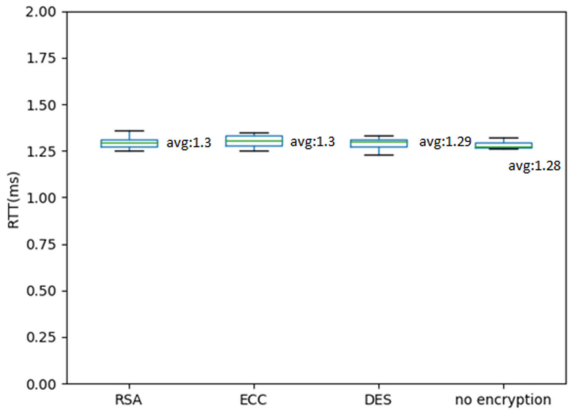


FIGURE 14. Result of the RTT experiment on the subsequent packets in the flow. It can be seen that there is no difference among different groups.

To minimize the impact of network conditions, our experiments are performed in the subnet described in Fig.12. Three Raspberry Pis are configured in the subnet, including an addressless server, an authenticated client, and a traditional server for comparison. ICMP ping is used to test RTT. The experiments are performed in four groups, the first one uses RSA for encryption, the second one uses ECC for encryption, the third one uses DES for encryption, and the last one does not use any encryption. The tests are repeated 10 times at different times independently in a day. Considering that our model only has an extra delay when establishing the connection, so only the first packet of the flow is used to measure the RTT. The results are shown in Fig.13.(a), Fig.13(b) and Fig.13(c).

The RTTs of the packets which are not the first packet of the connection are also tested. The result is shown in Fig.14. It can be seen that when the connection is established, there is no RTT difference among different groups.

Fig.13. and Fig.14. show that the additional overhead of DES is less than 1ms. In most network environments, it is much smaller than network delay. It does not affect network performance. The 1024-bit RSA brings more than 100ms extra delay, and the 192-bit ECC brings more than 20ms extra delay. This will affect the RTT in the connection establishment, but it has no effect on the subsequent messages. As a result, in a real network, this does not have a significant impact on user experience.

Considering that the RTT includes the overhead of client-side encryption and the server-side verification, the experiments for the encryption process and decryption process are then conducted. The experiments are repeated 10 times as well. The results are shown in Fig.15. Fig.15.(a) shows the encryption time of DES, RSA, and ECC, while Fig.15.(b) shows the verification time.

Although the overhead of RSA is large, the server-side processing time is only about 2ms, which is relatively small. This means that the server can handle multiple requests from different clients. ECC has a smaller encryption time, but it has a larger decryption time. However, DES has a smaller processing overhead. When it is unnecessary to use asymmetric encryption, DES is more appropriate.

The experiment on bandwidth is conducted in the same experimental environment. Here Iperf is used to measure bandwidth. The result is shown in Fig.16.

It can be seen that in our model, there is no difference in bandwidth no matter in the DES, RSA, ECC, or no encryption case.

E. THREATS-TO-VALIDITY OF THE EXPERIMENTS

The experiment results show that our model does not affect the RTT of the subsequent messages in a flow and the bandwidth. This is consistent with the analysis. This result does not change due to changes in the network environment and hardware platform.

The RTT of the first packet will be affected by the model. The additional RTT is affected by the CPU, network card, and the algorithm implementation. Special hardware module can be designed to conduct the encryption if necessary, which can greatly reduce the processing time and resource consumption in the verification. The network condition does not affect the additional time of the connection establishment. However, when the network condition is poor, the additional time of our model will be less obvious to the users. At the same time, the network environment, the hardware, and the software will not affect the security features of our model.

From the perspective of the threats to external validity, the experiment results are applicable to other Internet environments. Our model is built under an independent subnet on the Internet, the server is implemented based on Debian-based Linux OS, the algorithm is implemented in C, which are the mainstream hardware and software platforms on the Internet. As a result, the experiment results are generally applicable to IoT servers and private Internet servers.

VII. FUTURE WORK

The proposed Addressless IoT server is a new model that is different from the traditional IPv6 servers. While this model can better protect server security, there are also other potential applications that can be supported via it in the future.

Firstly, the encryption algorithm can be further improved. For example, we can further devise and implement encryption algorithms that consume fewer resources while ensuring security at the same level. The model can also be modified to be applied in specific IPv6 scenarios, such as 6LowPAN network.

Lastly, the mechanism in this paper is designed for private Internet servers. However, it can also be applied to public servers providing services to unspecified users. Furthermore, the idea that allocating a prefix to a device and allowing the device to use all the addresses under the prefix can not only work for servers but also clients. Some work has been done on these ideas. These results will be presented in the subsequent papers.

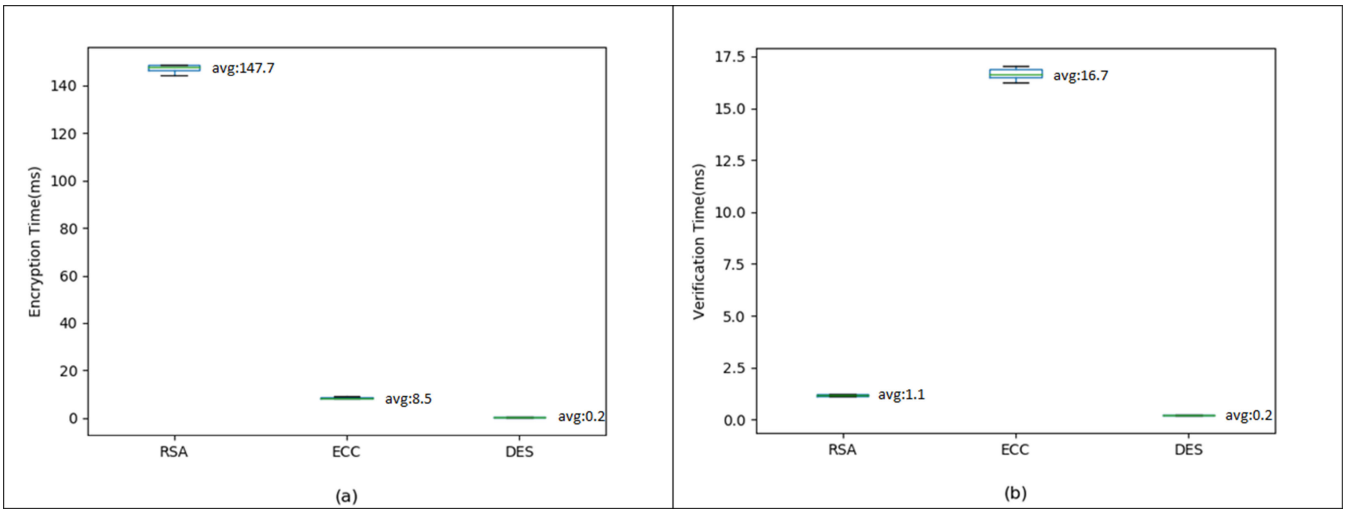


FIGURE 15. The encryption time and the verification time of different encryption algorithms. (a) is the result of the encryption time, and (b) is the result of the verification time.

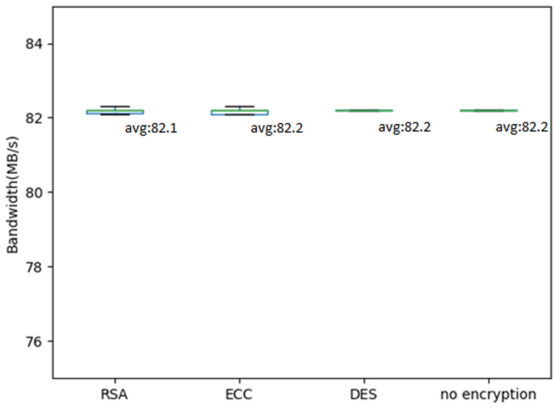


FIGURE 16. Result of the bandwidth experiment. It can be seen that there is no difference among different groups.

VIII. CONCLUSION

In this paper, a new IoT server model named addressless server is introduced. The model uses the prefix delegation mechanism, which allocates an IPv6 prefix instead of an IPv6 address to each server. The server listens on all addresseses under the prefix. Only authenticated clients are able to generate legitimate destination addresses using encryption, and the server verifies the data flow using the destination address.

The model uses the large IPv6 address space to hide the addresses in use. The server no longer uses a fixed IPv6 address. In this way, the one-to-one correspondence between the server and the IP address is eliminated, which makes it difficult for an attacker to find the correct address to launch attacks, so that the server can be protected from being scanned or attacked. Using these features, server security is guaranteed. The model is compatible with other network security protocols including IPsec, TLS, DTLS, and other IoT security solutions such as SSATP. The model has a wide range of application scenarios and can be widely used in various environments, such as the home IoT, etc.

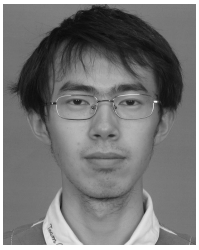
A prototype of the addressless IoT server model is implemented, and simulations and experiments are conducted based on the prototype. The simulation and experiment results demonstrate that the addresses generated by the algorithm can be used in verification, making the legal packets correctly responded while the unauthenticated devices are not being able to perceive the server. The generated addresses are random enough with a uniform distribution, which can be prevented from big data analysis. The results also show that the model has good performance features. It only brings a negligible additional delay during the connection establishment, and it has no effect on the delay of the subsequent data packets and the bandwidth.

REFERENCES

[1] S. Deering and R. Hinden, *Internet Protocol, Version 6 (IPv6) Specification*, document IETF RFC 1883, 1995.
[2] S. Deering and R. Hinden, *Internet Protocol, Version 6 (IPv6) Specification*, document IETF RFC 8200, 2017.
[3] T. Narten, *Neighbor Discovery for IP Version 6*, document IETF RFC 4861, 2007.
[4] R. Minerva, A. Biru, and D. Rotondi, "Towards a definition of the Internet of Things (IoT)," *IEEE Internet Initiative*, vol. 1, no. 1, pp. 1–86, May 2015.
[5] D. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast Internet-wide scanning and its security applications," in *Proc. 22nd USENIX Secur. Symp.*, 2013, pp. 605–620.
[6] K. Seo and S. Kent, *Security Architecture for the Internet Protocol*, document IETF RFC 5246, 2008.
[7] Y. H. Hwang, "IoT security & privacy: Threats and challenges," in *Proc. 1st ACM Workshop IoT Privacy, Trust, Secur. (IoTPTS)*, 2015, p. 1.
[8] N. Kushalnagar, *IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals*, document IETF RFC 4919, 2007.
[9] G. Glissa and A. Meddeb, "6LowPSec: An end-to-end security protocol for 6LoWPAN," *Ad Hoc Netw.*, vol. 82, pp. 100–112, Jan. 2019.

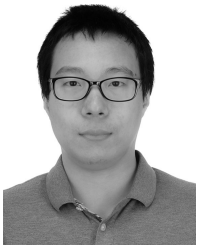
[10] S. Raza, S. Duquennoy, T. Chung, D. Yazar, T. Voigt, and U. Roedig, "Securing communication in 6LoWPAN with compressed IPsec," in *Proc. Int. Conf. Distrib. Comput. Sensor Syst. Workshops (DCOSS)*, Jun. 2011, pp. 1–8.
[11] L. Oliveira, J. Rodrigues, A. de Sousa, and J. Lloret, "A network access control framework for 6LoWPAN networks," *Sensors*, vol. 13, no. 1, pp. 1210–1230, 2013.
[12] Z. Shelby, K. Hartke, and C. Bormann, *The Constrained Application Protocol (CoAP)*, document IETF RFC 7252, 2014.
[13] S. Raza, H. Shafagh, K. Hewage, R. Hummen, and T. Voigt, "Lithe: Lightweight secure CoAP for the Internet of Things," *IEEE Sensors J.*, vol. 13, no. 10, pp. 3711–3720, Oct. 2013.
[14] E. Rescorla and N. Modadugu, *Datagram Transport Layer Security Version 1.2*, document IETF RFC 6347, 2012.
[15] F. Bonomi, "Fog computing and its role in the Internet of Things," in *Proc. 1st MCC Workshop Mobile Cloud Comput.*, 2012, pp. 13–16.
[16] A. Venčkauskas, N. Morkevicius, V. Jukavičius, R. Damaševičius, J. Toldinas, and Š. Grigaliūnas, "An edge-fog secure self-authenticable data transfer protocol," *Sensors*, vol. 19, no. 16, p. 3612, 2019.
[17] A. Venčkauskas, N. Morkevicius, K. Bagdonas, R. Damaševičius, and R. Maskeliūnas, "A lightweight protocol for secure video streaming," *Sensors*, vol. 18, no. 5, p. 1554, 2018.
[18] A. Ghafoor, M. Sher, M. Imran, and K. Saleem, "A lightweight key freshness scheme for wireless sensor networks," in *Proc. 12th Int. Conf. Inf. Technol. New Generat.*, Apr. 2015, pp. 169–173.
[19] A. Anand, M. Conti, P. Kaliyar, and C. Lal, "TARE: Topology adaptive re-kEying scheme for secure group communication in IoT networks," *Wireless Netw.*, vol. 26, no. 4, pp. 2449–2463, May 2020.
[20] S. Tomanson, T. Narten, and T. Jinmei, *IPv6 Stateless Address Autoconfiguration*, document IETF RFC 4862, 2007.
[21] T. Mrugalski, *Dynamic Host Configuration Protocol for IPv6 (DHCPv6)*, document IETF RFC 8415, 2018.
[22] R. Hinden and S. Deering, *IP Version 6 Addressing Architecture*, document IETF RFC 4291, 2006.
[23] F. Gont, *A Method for Generating Semantically Opaque Interface Identifiers with IPv6 Stateless Address Autoconfiguration (SLAAC)*, document IETF RFC 7217, 2014.
[24] R. Droms, Ed., *Dynamic Host Configuration Protocol for IPv6 (DHCPv6)*, document IETF RFC 3315, 2003.
[25] C. Huitema, T. Mrugalski, and S. Krishnan, *Anonymity Profiles for DHCP Clients*, document IETF RFC 7844, 2016.
[26] R. Rivest, *The MD5 Message-Digest Algorithm*, document IETF RFC 1321, 1992.
[27] J. Ullrich and E. Weippl, "Privacy is not an option: Attacking the IPv6 privacy extension," in *Proc. Int. Symp. Recent Adv. Intrusion Detection*, 2015, pp. 448–468.
[28] F. Gont and T. Chown, *Network Reconnaissance in IPv6 Networks*, document IETF RFC 7707, 2016.
[29] A. Cooper, F. Gont, and D. Thaler, *Security and Privacy Considerations for IPv6 Address Generation Mechanisms*, document IETF RFC 7721, 2016.
[30] O. Gasser, "Evaluating network security using Internet-wide measurements," Ph.D. dissertation, Technische Universität Munchen, Münich, Germany, 2019.
[31] D. Plonka and A. Berger, "Temporal and spatial classification of active IPv6 addresses," in *Proc. ACM Conf. Internet Meas. Conf. (IMC)*, 2015, pp. 509–522.
[32] F. Li, J. Yang, X. Wang, T. Pan, C. An, and J. Wu, "Characteristics analysis at prefix granularity: A case study in an IPv6 network," *J. Netw. Comput. Appl.*, vol. 70, pp. 156–170, Jul. 2016.
[33] T. Fiebig, "Something from nothing (There): Collecting global IPv6 datasets from DNS," in *Proc. Int. Conf. Passive Act. Netw. Meas.* Cham, Switzerland: Springer, 2017, pp. 30–43.
[34] K. Borgolte, S. Hao, T. Fiebig, and G. Vigna, "Enumerating active IPv6 hosts for large-scale security scans via DNSSEC-signed reverse zones," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2018, pp. 770–784.
[35] T. Fiebig, "In rDNS we trust: Revisiting a common data-source's reliability," in *Proc. Int. Conf. Passive Act. Netw. Meas.* Cham, Switzerland: Springer, 2018, pp. 131–145.
[36] R. Beverly, R. Durairajan, D. Plonka, and J. P. Rohrer, "In the IP of the beholder: Strategies for active IPv6 topology discovery," in *Proc. Internet Meas. Conf.*, Oct. 2018, pp. 308–321.
[37] J. P. Rohrer, B. LaFever, and R. Beverly, "Empirical study of router IPv6 interface address distributions," *IEEE Internet Comput.*, vol. 20, no. 4, pp. 36–45, Jul. 2016.
[38] O. Gasser, Q. Scheitle, S. Gebhard, and G. Carle, "Scanning the IPv6 Internet: Towards a comprehensive hitlist," 2016, *arXiv:1607.05179*. [Online]. Available: http://arxiv.org/abs/1607.05179
[39] P. Foremski, D. Plonka, and A. Berger, "Entropy/IP: Uncovering structure in IPv6 addresses," in *Proc. ACM Internet Meas. Conf. (IMC)*, 2016, pp. 167–181.
[40] J. Ullrich, P. Kieseberg, K. Krombholz, and E. Weippl, "On reconnaissance with IPv6: A pattern-based scanning approach," in *Proc. 10th Int. Conf. Availability, Rel. Secur.*, Aug. 2015, pp. 186–192.
[41] Z. Zuo, "Prediction algorithm of active IPv6 address prefix," *J. Commmunications*, vol. 39, no. Z1, pp. 7–14, 2018.
[42] A. Murdock, F. Li, P. Bramsen, Z. Durumeric, and V. Paxson, "Target generation for Internet-wide IPv6 scanning," in *Proc. Internet Meas. Conf.*, Nov. 2017, pp. 242–253.
[43] Z. Liu, Y. Xiong, X. Liu, W. Xie, and P. Zhu, "6Tree: Efficient dynamic discovery of active addresses in the IPv6 address space," *Comput. Netw.*, vol. 155, pp. 31–46, May 2019.
[44] K. Fukuda and J. Heidemann, "Who knocks at the IPv6 door?: Detecting IPv6 scanning," in *Proc. Internet Meas. Conf.*, Oct. 2018, pp. 231–237.
[45] D. Plonka and A. Berger, "KIP: A measured approach to IPv6 address anonymization," 2017, *arXiv:1707.03900*. [Online]. Available: http://arxiv.org/abs/1707.03900
[46] A. Judmayer, J. Ullrich, G. Merzdovnik, A. G. Voyiatzis, and E. Weippl, "Lightweight address hopping for defending the IPv6 IoT," in *Proc. 12th Int. Conf. Availability, Rel. Secur.*, Aug. 2017, pp. 1–10.
[47] M. Dunlop, S. Groat, W. Urbanski, R. Marchany, and J. Tront, "The blind Man's bluff approach to security using IPv6," *IEEE Secur. Privacy Mag.*, vol. 10, no. 4, pp. 35–43, Jul. 2012.
[48] O. Troan and R. Droms, *IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) Version 6*, document IETF RFC 3633, 2003.
[49] J. Brzozowski and G. V. D. Velde, *Unique IPv6 Prefix per Host*, document IETF RFC 8273, 2017.
[50] B. Hinden and B. Haberman, *Unique Local IPv6 Unicast Addresses*, document IETF RFC 4193, 2005.
[51] B. Claise, *Cisco Systems NetFlow Services Export Version 9*, document IETF RFC 3954, 2004.
[52] E. Rescorla, *HTTP Over TLS*, document IETF RFC 2818, 2000.
[53] D. Eastlake and P. Jones, *US Secure Hash Algorithm 1 (SHA1)*, document IETF RFC 3174, 2001.
[54] *Data Encryption Standard*, Federal Inf. Process. Standard, National Bureau of Standards, Gaithersburg, MD, USA, 1977.
[55] J. Daemen and V. Rijmen, "AES proposal: Rijndael," Banksys/Katholieke Univ., Leuven, Belgium, Tech. Rep., Jun. 1998. [Online]. Available: https://csrc.nist.gov/csrc/media/projects/cryptographic-standards-and-guidelines/documents/aes-development/rijndael-ammended.pdf
[56] D. E. Denning, "Digital signatures with RSA and other public-key cryptosystems," *Commun. ACM*, vol. 27, no. 4, pp. 388–392, Apr. 1984.
[57] Corporate Nist, "The digital signature standard," in *Proc. Commun. ACM*, 1992, pp. 36–40
[58] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ECDSA)," *Int. J. Inf. Secur.*, vol. 1, no. 1, pp. 36–63, Aug. 2001.
[59] C. Parr and J. Pelzl, *Understanding Cryptography: A Textbook for Students and Practitioners*. Berlin, Germany: Springer, 2012.
[60] T. Narten, R. Draves, and S. Krishnan, *Privacy Extensions for Stateless Address Autoconfiguration in IPv6*, document IETF RFC 4941, 2007.
[61] D. M'Raihi, *TOTP: Time-Based One-Time Password Algorithm*, document IETF RFC 6238, 2011.
[62] J. Robinson and M. Devarakonda, "Data cache management using frequency-based replacement," in *Proc. ACM SIGMETRICS Conf. Meas. Modeling Comput. Syst.*, 1990, pp. 134–142.
[63] T. Dierks and E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.2*, document IETF RFC 5246, 2008.
[64] R. Houseley, *Internet X.509 Public Key Infrastructure Certificate and CRL Profile*, document IETF RFC 2459, 1999.
[65] P. Nikander, J. Kempf, and E. Nordmark, *IPv6 Neighbor Discovery (ND) Trust Models and Threats*, document IETF RFC 3756, 2004.
[66] J. N. Goel and B. M. Mehtre, "Dynamic IPv6 activation based defense for IPv6 router advertisement flooding (DoS) attack," in *Proc. IEEE Int. Conf. Comput. Intell. Comput. Res.*, Dec. 2014, pp. 1–5.
[67] S. Thomson, T. Narten, and T. Jinmei, *IPv6 Stateless Address Autoconfiguration*, document IETF RFC 2462, 1998.

[68] R. Gagliano, S. Krishnan, and A. Kukec, *Certificate Profile and Certificate Management for SEcure Neighbor Discovery (SEND)*, document IETF RFC 6494, 2012.
[69] A. AlSa'deh and C. Meinel, "Secure neighbor discovery: Review, challenges, perspectives, and recommendations," *IEEE Secur. Privacy Mag.*, vol. 10, no. 4, pp. 26–34, Jul. 2012.
[70] E. Levy-Abegnoli, *IPv6 Router Advertisement Guard*, document IETF RFC 6105, 2011.
[71] F. Gont, *Implementation Advice for IPv6 Router Advertisement Guard (RA-Guard)*, document IETF RFC 7113, 2014.



RENJIE LIU received the B.S. degree in electronics engineering from Tsinghua University, Beijing, China, in 2013, where he is currently pursuing the Ph.D. degree with the Department of Electronic Engineering.

His research interest includes network security, network measurements, and IPv6 networks.

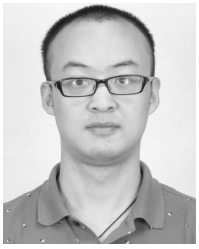


ZHE WENG received the B.S. degree in microelectronic science and engineering from Tsinghua University, Beijing, China, in 2017, where he is currently pursuing the M.S. degree in information and communication engineering.

His research interests include the area of the transition from IPv4 to IPv6, the Internet of Things (IoT), and network security.

SHANSHAN HAO received the B.S. degree in microelectronic science and engineering from Tsinghua University, Beijing, China, in 2017, where she is currently pursuing the M.S. degree in information and communication engineering.

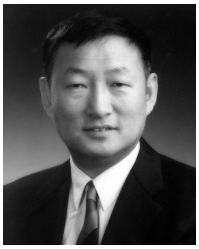
Her research interest includes improving domain name system security based on blockchain technology and IPv6 development.



DELIANG CHANG received the master's degree in engineering from Tsinghua University, Beijing, China, in 2013, where he is currently pursuing the Ph.D. degree.

His research interests include network measurement and network security.

CONGXIAO BAO is currently an Associate Professor with the Research Institution of Network Science and Cyberspace Technology, Tsinghua University. She has 25 years of research experiences on network architecture, protocol, network measurements, and network management. She has been the authors of nine IETF RFCs and more than 40 articles and more than 35 granted Chinese patents. She is also the Steering Committee Member of APAN Medical WG.



XING LI received the B.S. degree in radio electronics from Tsinghua University, Beijing, China, in 1982, and the M.S. and Ph.D. degrees in electrical engineering from Drexel University, USA, in 1985 and 1989, respectively.

He is currently a Professor with the Electronic Engineering Department, Tsinghua University. He is the Deputy Director of the China Education and Research Network (CERNET) Center. He published more than 300 articles in his research areas and the coauthor of 11 IETF RFCs. His research activities and interests include compute networks, multimedia communications, and statistical signal processing. He was a member of Internet Architecture Board (IAB), the Co-Chair of the Coordinating Committee for Intercontinental Research Networking (CCIRN), the Chair of the Asia pacific Network Group (APNG), a member of the APNIC Executive Council, and the Chair of Internet Sub Chapter of Computer Society of China (CCF).