Journal Pre-proof

Am I Eclipsed? A Smart Detector of Eclipse Attacks for Ethereum

Guangquan Xu, Bingjiang Guo, Chunhua Su, Xi Zheng, Kaitai Liang, Duncan S. Wong, Hao Wang

PII: S0167-4048(18)31379-8
DOI: https://doi.org/10.1016/j.cose.2019.101604
Reference: COSE 101604

To appear in: *Computers & Security*

Received date: 30 November 2018
Revised date: 11 August 2019
Accepted date: 1 September 2019

Please cite this article as: Guangquan Xu, Bingjiang Guo, Chunhua Su, Xi Zheng, Kaitai Liang, Duncan S. Wong, Hao Wang, Am I Eclipsed? A Smart Detector of Eclipse Attacks for Ethereum, *Computers & Security* (2019), doi: https://doi.org/10.1016/j.cose.2019.101604

Am I Eclipsed? A Smart Detector of Eclipse Attacks for Ethereum

Guangquan Xu[a,b], Bingjiang Guo[a], Chunhua Su[c], Xi Zheng[d], Kaitai Liang[e,*], Duncan S. Wong[f], Hao Wang[g]

[a] *Tianjin Key Laboratory of Advanced Networking (TANK), College of Intelligence and Computing, Tianjin University, P.R. China*
[b] *Big Data School, Qingdao Huanghai University, P.R. China*
[c] *University of Aizu, Aizuwakamatsu, Japan*
[d] *Department of Computing, Macquarie University, Australia*
[e] *Department of Computer Science, University of Surrey, U.K.*
[f] *CryptoBLK, Hong Kong*
[g] *Department of Computer Science, Norwegian University of Science and Technology, Norway*

Abstract

Blockchain security has been drawing a tremendous attention from industry and academic due to its prevalence on real-world applications in these years, such as distributed blockchain-based storage systems. Since being deployed in distributed and decentralized network, blockchain applications may be vulnerable to various types of network attacks. This paper deals with "eclipse attacks" enabling a malicious actor to isolate a system user by taking control of all outgoing connections. Although being known from practical blockchain applications, eclipse attacks, so far, are hard to be detected. To solve this problem, this paper designs an eclipse-attack detection model for Ethereum platform, ETH-EDS, based on random forest classification algorithm. Specifically, via the collection and investigation over the normal and attack data packets (across the network), we find out that the information in the attack packets includes the tags *packets_size, access_frequencies* and *access_time*, which may help us effectively detect the attack. After training the data packets which we collect from the network, our ETH-EDS is able to detect malicious actor with high proba-

*Corresponding author
Email address: k.liang@surrey.ac.uk (Kaitai Liang)

bility. Our experimental analysis presents evidence to show that the detection of malicious network node (i.e., the malicious actor) is with high accuracy.

Keywords: blockchain security, eclipse attacks, detection, malicious actor, random forest classification.

1. Introduction

The Ethereum, nowadays, has emerged as one of the most successful cryptocurrency platforms. Compared to Bitcoin [1], Ethereum has been seen as the blockchain 2.0 that is able to extend the distributed ledger technology to support various decentralized applications via more flexible layers and security mechanisms, like access control and Ethereum smart contracts which can be written in a turing-complete language. Although being more advanced than the design of Bitcoin, Ethereum nodes (much like those in Bitcoin) also need to exchange the state of ledger in a peer-to-peer network via emulating a structured graph based on the Kademlia DHT [2].

Recent research works [3, 4] have concluded that the security of a proof-of-work blockchain relies on the security of its underlying peer-to-peer network. In other words, if the peer-to-peer network is partitioned and it nodes have different copies of information about the ledger, these nodes will not come to any agreement on the unique ledger across the network. This potential risk is emphasized in [5] by defining and demonstrating the first eclipse attack on the Bitcoin peer-to-peer network. In the attack, a malicious actor (for simplicity, hereafter we refer to malicious actor as attacker) can completely control its target's access to other network nodes, so that the attacker is able to block the target's view of the ledger, and even further make use of the target's computing resource for more sophisticated attacks. Eclipse attack can be used as a useful building block for other attacks to blockchain platform [6]. The work [5] shows that eclipse attacks can be useful in affecting the mining power of an attack target (hereafter we refer to target as victim) so as to attack the blockchain underlying consensus algorithm. [6] further states that Ethereum smart contract

may be vulnerable in a case where nodes receive inconsistent information of ledger. Meanwhile, Eclipse attacks can be leveraged as parts of adversarial strategies for injecting the inconsistency. Later on, Yuval Marcus [2] proves that the same attack also affects the Ethereum peer-to-peer network to make the platform unstable.

Although being defined and studied in Bitcoin and Ethereum, this type of attacks has not been effectively detected yet. In this paper, we propose a detection model based on machine learning technology to protect Ethereum network clients from the eclipse attacks. When trying to "eclipse" an honest node, an attacker needs to send continual connection requests to the target. Too many unsolicited incoming connections from the attacker will occupy all incoming connections to the node, which leads to a consequence that the honest node cannot receive information of the current ledger from others. In addition to the above attacking mode, the attacker may also send the *ping* to the target repeatedly with the carefully-crafted set of node identifiers, which can populate the target's *table, i.e.* the routing table. In this paper, we find out that both of the attacking approaches may leak information for us to trace the attacks. We propose a more pervasive and practical eclipse-attack model based on the eclipse attacks defined in [1]. This model can examine how the target's states change during a complete eclipse attack process. Normal and attack packets can be collected according to the states change in the attack and further, they are intaken by machine learning algorithms for detection analysis. We select several features of the packets the target receives, and then train a detection model to identify the type of new packets [7].

The next of the paper is organized as follows. Section 2 provides an introduction to the Ethereum network and the eclipse-attack model that we build from the two eclipse-attack methods. Section 3 presents our detection model based on random forest classification. Section 4 shows the experimental results of the detection model w.r.t. the eclipse attacks. Section 5 introduces the existing related works. Section 6 and section 7 give the conclusions and the discussion of future work.

2. Eclipse Attack on Ethereum P2P Network

In this section, we will deal with a detailed investigation of eclipse attack on the Ethereum peer-to-peer network based on the Geth version 1.6.6 client. Geth is the command line interface for running a full Ethereum node implemented in the Go language. The main neighbour discovery protocol of Geth prior to v1.8.0 is RLPx Node Discovery Protocol v4. This peer-to-peer protocol results in critical eclipse-attack vulnerabilities.

2.1. Ethereum P2P network

Peer identification. A peer in an Ethereum network is identified by the nodeID, which is a 512-bit cryptographic ECDSA public key. It is easy for one to leverage a computer to launch multiple Ethereum nodes with different nodeIDs. For a node, what it usually needs to do is to run the ECDSA key generation algorithm [1]. The algorithm, however, will not check if the nodeID corresponds to a unique network address. Therefore, one can theoretically run the unlimited number of nodes on a number of machines with the same IP address.

Network Connection. In an Ethereum peer-to-peer network connection, UDP is used to find other peers and further establish a connection channel to exchange ledger information [8]. All ledger information is transported through encrypted and authenticated TCP connection. UDP connection can be made up to 16 concurrently, and the total number of UDP connection is unlimited. The limit of TCP connection is *Maxpeers*, which is made to 25 by default. There are 2 groups of UDP message. A *ping* message solicits a *pong* message in return [9]. This pair of messages is used to check if a neighbour node is alive/active. A *findnode* message solicits a *neighbour* message that contains a list of 16 nodes that have been found by the responding node. A node will only respond to a *findnode* request when the querying node is already in his *db*, which is a type of network information storage. To eclipse a node by monopolizing connections, an attacker must repeatedly occupy all *Maxpeers* of the victim's TCP connections [4]. A TCP connection can be outgoing only if the client allows/sets it in the beginning;

otherwise, it will be set as incoming connections by default. The outgoing TCP connections can be initiated up to $\frac{1}{2}(1 + Maxpeers)$ with other peers by the client. However, there is no limit on the number of unsolicited incoming TCP connections, prior to Geth v1.8, other than *Maxpeers*. This means that a node's *Maxpeers* of its TCP connections can all be the unsolicited incoming connections.

Network Information Storage. Neighbour nodes' information is stored into two data structures. The *db* is a long-term database which stores node information that a client has been bonded. A node will be bonded if it returns a valid and corresponding pong response after it receives the *ping* message. There is no limit on the size of *db*. Each *db* entry consists of a nodeID, IP address, TCP port, UDP port, time of the last *ping* sent from the node, time of the last pong received at the node, and the number of times the node failed to respond to a *findnode* message. A node's age is the time elapsed since the last pong received from the node. Each hour the client runs an eviction process to remove nodes from the *db* which are there longer than 24 hours.

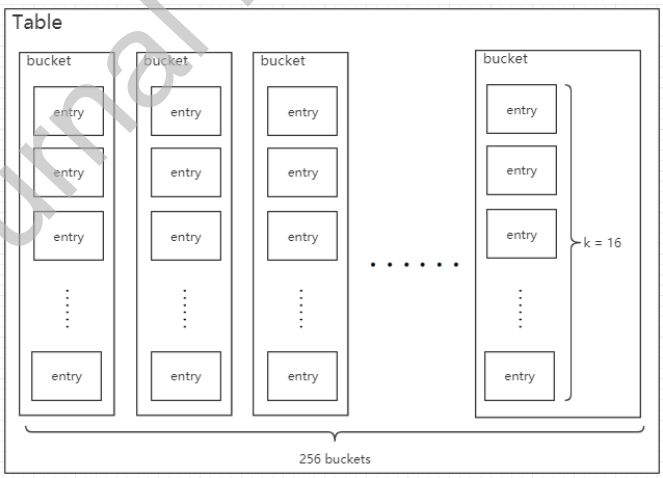

Fig. 1. The structure of Table storage

There are 256 buckets in the *table*, in which each of the bucket consists of

$n = 16$ entries. Fig. 1 illustrates the Network Information Storage. Each entry stores Ethereum node information including nodeID, IP address, TCP port and UDP port. When a new node is added into a bucket, it's mapped by *logdist* function which is a modification of the XOR metric in the Kademlia protocol. The *logdist* function is used to measure the distance between two nodeIDs. First, each nodeID is hashed to a 256-bit value with the SHA3 algorithm. If the r most significant bits of the two hash values are identical, but the $r + 1st$ bit is different, we then say that the *logdist* of this node is r. A nodeID that has *logdist* r is mapped to the bucket $256 - r$ of the client's *table*.

2.2. Eclipse-attack methods

To eclipse a node, an attacker has to populate the data structures of the victim, which makes the victim only obtain ledger information through the attacker's nodes [1].

Bonding. The most common way to populate both the *db* and the *table* is to launch the bonding process. When the client bonds with a node, the clients first checks if

(1) the node exists in his *db*.
(2) the *db* records 0 failed responses to *findnode* requests, and
(3) the *db* records that the node has responded with a pong within the last 24 hours.

If all the above checks pass, the client tries to add the node to its *table*. Note that the node can be added in the *table* if there is still space in the table. Otherwise, the client sends a *ping* to the node to check if it's alive. The bonding is made successfully, if the node responds with a *pong*. If there is a success bonding, the client updates the node's entry in its *db* and its *table*, simultaneously.

Unsolicited pings. The client receives unsolicited *ping*s from other nodes, respond with a *pong* message and further bonds to the node successfully.

Lookup. The iterative *lookup*(t) method can be used for clients to discover new nodes. The *lookup*(t) method relies on a notion of "closeness" to a target node t. Much like the Kademlia protocol, the "closeness" is defined by bitwise XOR between two nodes. The client sends *findnode* containing target node t to closest 16 nodes in the *table*. The nodes receiving *findnode* will query its *table*, and send closest 16 nodes in the *table* to its sender. Accordingly the client has the information of the $16 * 16 = 256$ nodes.

Selecting connection peers. A task runner is initiated and operated continuously when an Ethereum client boots up. The task runner creates up to $b = \frac{1}{2}(1 + Maxpeers)$ (by default 13) outgoing TCP connections to other nodes in the Ethereum network [10]. In general, a half of outgoing TCP connection peers are selected from *look_up* and another half from its *table*. The *random_buffer* can hold $\frac{1}{2}(\frac{1}{2}(1 + Maxpeers))$ nodes. During the task_creation stage, the client node selects peers from the *table* at random to fill *random_buffer*.

2.3. Eclipse Attack Model

Generally speaking, there are two methods to launch eclipse attacks on Ethereum network. One is that an attacker eclipses an Ethereum victim by establishing *Maxpeers* incoming TCP connections to its own malicious nodes before the client is going to establish any outgoing TCP connections, and the other one is to eclipse by owning the *table*. When rebooting, the victim occupies all thirteen of her outgoing connections to its own adversarial nodes with high probability [11]. To complete the eclipse, the attacker monopolizes the remaining connections of the victim with unsolicited incoming connections immediately. Based on the attack methods, we design an eclipse-attack framework from the view of victim. We define 4 states for node in the framework, and via the change of the states we can tell if a node is under eclipse attacks at that very moment [12]. The states are defined as follows.

Running. This state means that the node has already lasted at least for twenty-four hours, and it has established peer connection. The databases of the

node, *db* and *table*, may have some peer information. *db* contains the node that responds to a *ping* message from the client with a *pong* message. And the *table* is filled with highly-skewed due to the feature of SHA3, because the SHA3 maps each nodeID to a random 512-bit string. The probability that the second string has its first r bits identical to that of the first string, and the $r+1st$ bits are different, is defined as [10].

$$p_r = \frac{1}{2^{r+1}} \tag{1}$$

Reborn. After a node reboots for some reason (e.g., crash-and-recover), it will change to a *reborn* state. We find out an important feature while the node is in this state. The *table* is always empty after the reboot. And this gives an attacker a window to attack the node in such a way that once the victim reboots, the attacker immediately initiates incoming connections or carefully-crafted packets to the victim. Note that this may be the best time for collecting the malicious packets [13].

Submerge. If the attacker establishes *Maxpeers* incoming TCP connections of the victim to its own adversarial nodes, we call this state *submerge.* At this state, all connections of the victim are forced to be set as incoming connections.

Poisoned. When the table of a victim is inserted much crafted nodeID by the attacker, we call it *poisoned.* The victim here forms all outgoing connections to the nodes of the attacker with high probability.

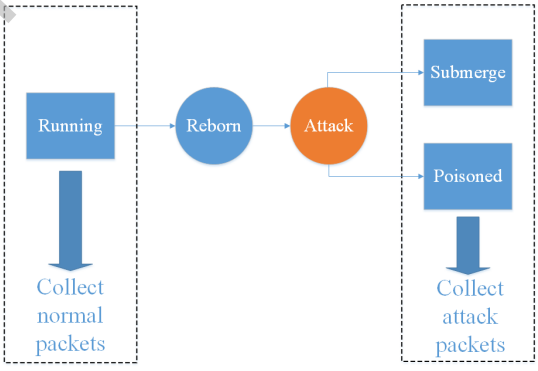

Fig. 2. Eclipse-attack model

Fig. 2 illustrates that the states change during an active eclipse attack. When a node A is active for a while, it's in the *running* state. If an attacker B wants to launch eclipse attacks to the node A, it needs to send many *ping* requests to A. Once the victim A changes into the *reborn* state, (as we say, it reboots for some reason) it will be eclipsed with high probability. Which the state, either *submerge* or *poisoned*, A goes into, will depend on its connection configuration. If A doesn't create outgoing TCP connections, it may switch to the *submerge* state. Otherwise, the table of A will be poisoned by a crafted message from the attacker B.

To train our detection model, we need to collect ground data beforehand [14]. Our victim is designed to be a Geth client (version 1.6.6). We firstly collect normal access connection packets in the *running* state and further we make an attack script that attempts to eclipse the victim by sending *ping* repeatedly. We start packets collection before the victim reboots and continuing to the moment when its incoming connections are fully occupied by us or its *table* is filled with our node entries [15].

3. Detection Model based on RFC

To detect the eclipse attacks, we construct a detection tool based on random forest classification. Recall that to monopolize or poison into a node, the attacker has to send continual connection requests to the victim. To play the role of a victim, we can collect all UDP packets from the unsolicited node [16, 17]. We can tell if the resources are honest according to the requests sequence which we receive. As analyzing the packet's data of eclipse attacks, we find that it has two "many-to-one" features: 1) The source and destination addresses of the data flow have a many-to-one relationship [18], 2) The source address of the data stream has a many-to-one relationship with the destination port. When attacking a service of the target host, the attacker sends a large amount of data flow to the fixed port of the target host [19].

The process of training the detection model is illustrated in Fig. 3. First,

we extract the feature vectors from the training UDP packets. Next, we train a random forest classifier using the training dataset, which combines the feature vectors and their corresponding labels. Finally, we test our model with the new UDP packets. To do so, we perform the same feature extraction process as the training dataset and feed the feature vector to make the final prediction. We use these predictions as the expected labels.

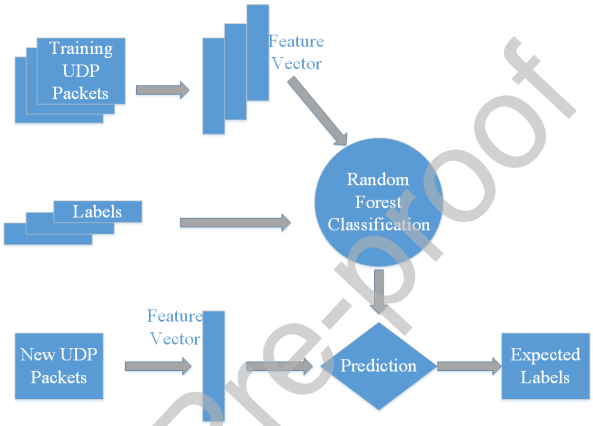

Fig. 3. The detection model training process

3.1. The Selection of Classification Features

Information entropy is a measure to describe variable uncertainty [20, 21]. In this paper, we consider to leverage information entropy of data stream to describe the characteristics of the eclipse attacks [22]. According to the definition, the information entropy of variable Y is calculated via the following Eq. (2), where *p(yi)* is the prior probability of variable Y.

$$H(Y) = -\sum_{i} p(y_i)\log_2(p(y_i)) \tag{2}$$

The information entropy of the variable Y about X is shown in the Eq. (3), where $p(y_i|x_j)$ is the posterior probability of y_i with respect to x_j

$$H(Y|X) = -\sum_{j} p(x_j)\sum_{i} p(y_j|x_i)\log_2(p(y_i|x_j)) \tag{3}$$

The "SourceIP", "DestinationIP" and "DestinationPort" are used to represent the source address, destination address and destination port of the data flow, respectively. The information entropy of SourceIP to DestinationIP (SIDI) and SourceIP to DestinationPort (SIDP) are used to characterize two kinds of features [20, 23]. The two information entropy constitutes the information entropy of data flow (DSIE), which reflects the uncertainty of SIDI and SIDP. We here take SIDI as an example, and it is defined in the Eq. (4). The calculation of DSIE is defined as follows. Suppose the total number of data streams sampled is S, and the set of source addresses in these data streams is $\{si_i|i = 1, 2, \cdots, N\}$, and the destination address set is $\{di_j|j = 1, 2, \cdots, M\}$. We define the vector $A[M]$ and the matrix $B[N][M]$, where A_j represents $A[j]$, the number of data streams whose destination address is di_j, and B_{ij} represents $B[i][j]$, the number of data streams whose source address is si_i and the destination address is di_j. According to the Eq. (3), we have

$$\begin{aligned} SIDI &= -\sum_{j} p(di_j) \sum_{i} p(si_i|di_j) \log_2(p(si_i|di_j)) \\ &= \sum_{j=1}^{M} \frac{A_j}{S} \sum_{i=1}^{N} \frac{B_{ij}}{A_j} \log_2(\frac{B_{ij}}{A_j}) \end{aligned} \tag{4}$$

In the following, we define other features which can be calculated directly from data packets and help us identify the eclipse attacks.

The *packets_size* represents the size of access packets. The main difference between normal access and attack packets is that the latter one has much more number of *ping-pong* than *findnode-neighbors*. And the size of these two kinds of UDP packets are different as well. From the above evidence, we may identify the feature of attack data.

The *access_frequencies* represents the node visit frequency. In order to eclipse a victim, an attacker needs to send *ping* request repeatedly. The busy traffic flow can be used as one of the features to examine the eclipse attacks.

The *access_time* represents the connection time. When a node cannot respond with a *pong* to its sender on time, the sender may cancel this connection

immediately. As for the attack scenario, the attacker, on the contrary, prefers to wait till the reply rather than giving up easily.

These statistical features are much simpler and more intelligible, while the information entropy approach is based on a destination set that contains many victims. To simplify our experiment, we make use of the above statistical features.

3.2. Random Forest Classification

The random forest classification is a machine learning algorithm which improves the detection accuracy without significant cost for computational complexity. Random forest can overcome several problems with decision trees, including a reduction in overfitting and less variance. It is used to establish a forest in a random way [24]. The forest is composed of a large number of decision trees. There is no correlation between the trees in the random forest. When a new input sample enters the forest, the decision trees are judged respectively, and the sample category is determined according to the decision. Combined with the characteristics and advantages of Bagging and random subspace algorithms, the random forest takes decision tree as the classifier to carry out training without putting backing sampling of Bagging algorithm. At the same time, the random subspace algorithm takes only part of the samples in the training set for training, and the results are determined by decision tree voting. In the random forest, each classification tree is a binary tree that follows the principle of recursion division. The binary tree divides the training set successively from the root node, and its generation follows the top-down principle. The root node that contains all training data is divided into left and right nodes according to the principle of minimum purity, and it contains a subset of training data. The node continues to split according to the same rule, and stops growing when the branch stop rule is satisfied. The implementation of the algorithm is designed as follows.

(1) To initialize the data set D, the Bootstrap method is used to randomly extract k sample sets each time from the data set, so as to generate k

classification trees.

(2) S variables are randomly obtained from the nodes of each tree in k classification trees [20]. The most representative variables are selected from these variables. The threshold of classification is determined by multiple classification points.

(3) Do not trim classified trees to make them grow indefinitely [25].

(4) The new samples are divided into the random forest through the constructed random forest, and the classification results are determined by the classifier vote.

3.3. RFC Model Training

Each node of the random tree in the RFC model can be considered as a weak classifier [11]. A classification criterion $h(x, \theta) \in \{0, 1\}$ is obtained by calculating the training sample set that reaches the node of Ω. Here we let $x \in R^m$ represent a training sample, $\theta \in \{\phi, \psi\}$ be the parameter of this weak classifier, $\phi(\psi)$ be a filter function, ψ be a column vector parameter or a matrix parameter, and θ determine the pattern of the classification hyperplane of the weak classifier.

a) The nonlinear classification [26] surface is defined in the Eq. (5), where $\phi(\cdot)$ is an indicator function. For the sample $x = (x_1, x_2, x_3) \in R^3$, we let $\phi(x) = (x_1, x_3, 1)^T$, $\psi = (w_1, w_2, \tau)$, and $h(\cdot)$ represent a classification surface.

$$h(x, \theta) = \delta(\phi^T(x)\psi_\phi(x) > 0) \tag{5}$$

b) Linear classification is defined in the Eq. (6), where ψ is a parameter matrix.

$$h(x, \theta) = \delta(\phi^T(x)\psi > 0) \tag{6}$$

When the sample is satisfied with $h(x, \theta) = 1$, it falls into the left subtree. This is regarded as a rational behaviour. Otherwise, it is regarded as irrational (falling into the right subtree).

It runs recursively until the number of sample falling is below the threshold or reaches the specified maximum depth. At the end of the recursion, this node

is called the leaf node [27]. The optimal coefficient θ^* can be found at each node, which makes the training sample achieve the best effect, as shown in the Eq. (7).

$$\theta^* = argmax IG_{\theta_j \in \Gamma_{sub}}(\theta_j|\Omega), \tag{7}$$

where Γ is the subset of complete parameter space Γ. Each node Γ_{sub} is randomly selected from Γ, which reflects the randomness in the process of node split. $IG(\cdot)$ represents information gain, being used to measure the decrease of sample impurity after splitting. It is defined as

$$IG(\theta|\Omega) = H(\Omega) - \sum_{i \in \{l,r\}} \frac{|\Omega_i(\theta)|}{\Omega} H(\Omega_i(\theta)) \tag{8}$$

We further let $\Omega = \{(x_i, y_i)\}_{i=1}^{N}$ represent the set of all samples that fall into the node (where $|\Omega| = N$), $\Omega_l(\theta)$ and $\Omega_r(\theta)$ represent the sample set that falls into the left and right child nodes under the argument θ, $H(\Omega)$ denote the impurity of the sample set that falls into a node. $H(\Omega)$ can be represented by information entropy shown in the Eq. (9).

$$H_{entropy}(\Omega) = -\sum_{c=1}^{N_c} p(c|\Omega) \log p(c|\Omega) \tag{9}$$

Our RFC model training process works as follows. In the model training process, we train n decision trees. We firstly get the sample set T' by resampling p samples from training set T using the function *sample_withResample*, and then we obtain the feature set Att. Secondly, we sample k features without replacement from the feature set Att. And T' remains the features which the Att' contains. Eventually, we can build the decision array DT.

3.4. Classification of RFC Models

After the RFC model training, the test sample x passes through each tree to reach a certain leaf node [28]. Therefore the probability that sample x belongs to c can be defined in the Eq. (10).

$$p(c|x) = \frac{1}{T} \sum_{t=1}^{T} p_t(c|x) \tag{10}$$

Algorithm 1: RFC model training process

Input: T,p,k

Output: DT

```
for i = 1; j ≤ n; i++ do
    T' = sample_withResample(T, p);
    Att = getAttributes(T');
    Att' = sample_withoutResample(Att, k);
    T" = remainAttributes(T', Att');
    DT[i] = createDecisionTree(T");
return DT;
```

where T is the number of random trees in the forest [19], and $p_t(c|x)$ is the category distribution of leaf nodes. The decision for the x category can be also defined as the Eq. (11).

$$c = argmax_{c\in\{1,\cdots,N_c\}} p_t(c|x) \tag{11}$$

The classification process of the RFC model is a majority voting process [16]. We define the test sample set as $E = \{e_1, e_2, \cdots, e_m\}$, the set of decision tree obtained by training as $DT = \{dt_1, dt_2, \cdots, dt_n\}$, the subscript array that records the classification results of each decision tree as CIR[n], the category set as $C = \{c_1, c_2, \cdots, c_n\}$, and the classification result set as $CR = \{CR_1, CR_2, \cdots, CR_m\}$, respectively. Our classification process of the RFC model is presented in the Algorithm 2.

4. Our Experiment

In this section, we present our experiment to highlight the accuracy and effectiveness of our detector of eclipse attacks to Ethereum network. The experiment mainly consists of four steps.

Step 1: Data collection. We firstly collect normal access connection packets during the *running* state and then we use attack script that attempts

Algorithm 2: The RFC model classification process

Input: DT,E

Output: CR

```
for i = 1; i ≤ m; i++ do
    for j = i; j ≤ n; j++ do
        CIR[j] = 0;
for j = 1; j <= n; j++ do
    classifyResultIndex = classify(E[i], DT[j])
    CIR[classifyResultIndex]++;
maxIndex = getMaxAppeared(CIR);
CR[i] = C[maxIndex];
return CR;
```

to eclipse a victim by sending *ping* repeatedly. We start to collect eclipse-attack packets before victim reboots till its incoming connections occupied by us or its *table* is filled by our node entries. At this step, we make use of *wireshark* to collect the UDP packets from the victim. The process to capture sample data by *wireshark* is shown in Fig. 4. We also add Ethereum devp2p protocol dissector plugin in *wireshark* to help us analyze the collected UDP packets.

Step2: Data pre-processing. The data collected is with 4 types including *ping*, *pong*, *findnode* and *neighbors*. The packet structure is described as follows[29].

(1). All packets are signed with ECDSA-secp256k1 keys (representing a node ID).

- For authenticity

- Signature: sign(privkey, sha3(packet-type || packet-data))

- 65-byte compact ECDSA signature containing the recovery ID as the last element.

- Please refer to the code [30] for more information on how NodeID is recovered from the signature.

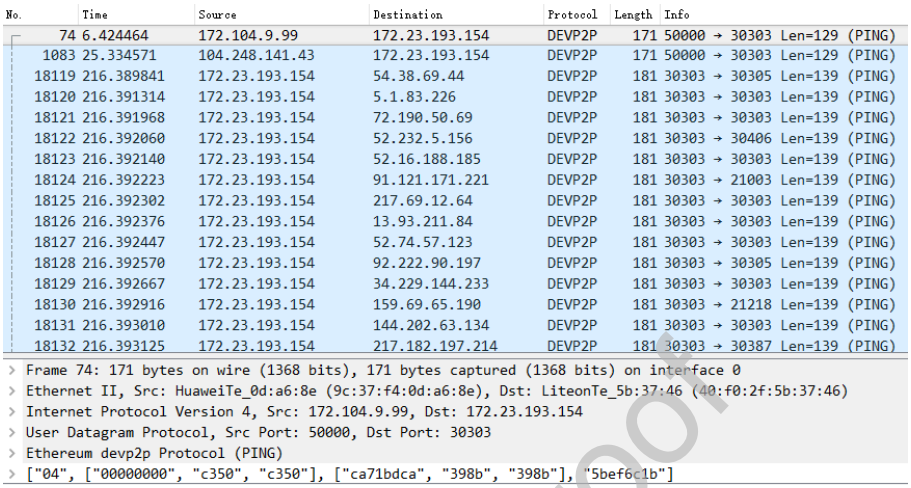

Fig. 4. The packets collected by Wireshark

(2). All packets are prepended with the SHA3-256 hash of the underlying data of the packet.

- For integrity
- Hash: sha3(signature || packet-type || packet-data)
- 32 bytes.

(3). Packet Type: Single byte $< 2 * *7$ // valid values are in $[1, 4]$.

(4). Full UDP Packet Payload: hash || signature || packet-type || packet-data [29].

At this stage, we use an Ethereum UDP packet dissector for discovery protocol v4 to decode a pcap file of captured Ethereum packets into a readable format.

The data decoded are illustrated in Fig. 5. Each packet has its packet type, source IP and destination IP. The sampling period of attack traffic increases from 5ms to 25ms, with a single increase by 5 ms, while the sampling period of background traffic is fixed at 5s. Among the five sets of data obtained, each set of data takes 1,000 samples continuously, and each group of 50 samples is a sample sequence, with a total of 20 sample sequences.

Step3: Training model. Firstly, we take a statistical analysis for a direct

```
----------------------------------------
Timestamp: 11/13/2018 8:19:43.504698PM
Packet #: 308
Source: 172.23.193.154
Destination: 104.248.208.204
Packet length: 191
Packet type: Pong
Packet signed by:
6dc1a23da3938f327ae075f3293e132b93942a5789e06a17f75b8558836c689d9fe077114aa362738e1d8e1d0ce0166a1fb03b58389a364d3ae2549a970cc0f5
Hash of packet: 386418ed73bd9d57840514386cd502e79d96b68738ec3233199eab821de8b53e
Pong to IP: 104.248.208.204 UDP: 30303 TCP: 30303
ReplyToK: 2ef430ab3f5d1532666d7b6c96f4c544b4c1ce6c4d231089b4ef6eef79581b88
Expiration: 2018-11-13 20:20:03 +0800 CST
----------------------------------------
Timestamp: 11/13/2018 8:19:43.505077PM
Packet #: 309
Source: 172.23.193.154
Destination: 104.248.208.204
Packet length: 181
Packet type: Ping
Packet signed by:
6dc1a23da3938f327ae075f3293e132b93942a5789e06a17f75b8558836c689d9fe077114aa362738e1d8e1d0ce0166a1fb03b58389a364d3ae2549a970cc0f5
Hash of packet: 65a7561a970c70cf0b58f1da80934da25abbc6ea871ed0cce86218591570db75
Ping to IP: 104.248.208.204 UDP: 30303 TCP: 0
Ping from IP: :: UDP: 30303 TCP: 30303
Expiration: 2018-11-13 20:20:03 +0800 CST
Version: 4
----------------------------------------
```

Fig. 5. The distributions of different packets size

view of UDP packets distribution in two states. They are presented as follows.

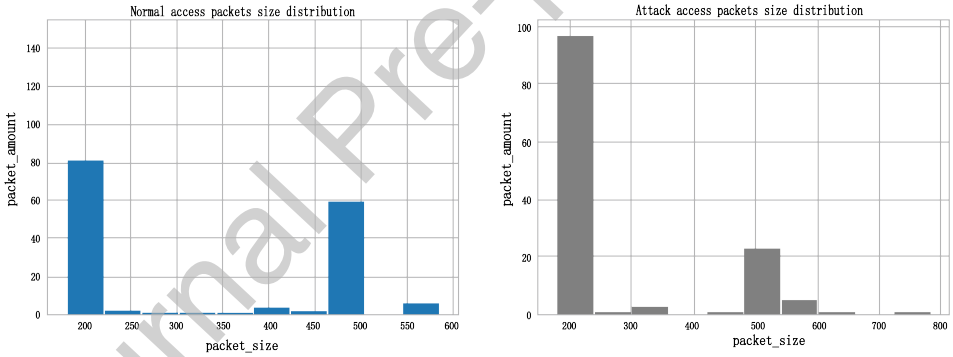

Fig. 6. The distributions of different packets size

As shown in Fig. 6, honest and malicious packets have different packets of size distribution. To eclipse an honest node, an attacker has to send many *ping* requests to the victim. The packets in the type of *ping, pong* will have less data information compared to those with the types of *findnode, neighbors*. Accordingly, their sizes have different distributions.

Fig. 7 shows that attack access consumes higher time complexity. This indicates that a normal access is built for short connection. When a victim cannot respond with a *pong* to an attacker on time, the attacker may stay

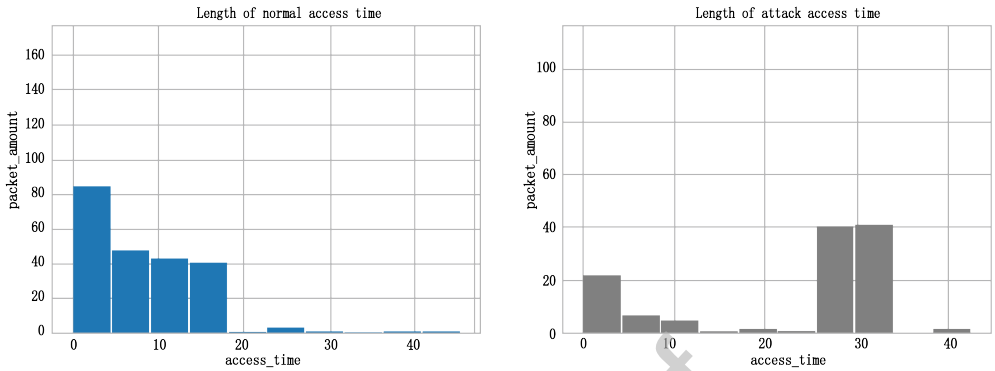

Fig. 7. The distributions of different request time

longer for waiting.

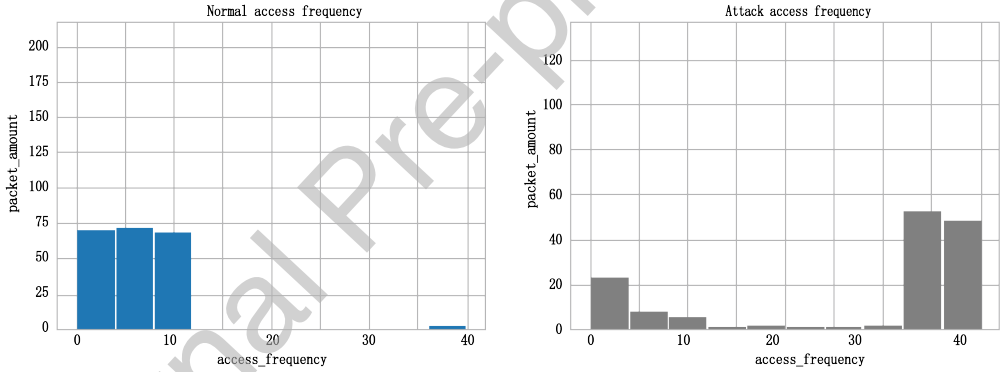

Fig. 8. The distributions of different access frequency

Fig. 8 depicts that there exists much higher visit frequency while a node is under eclipse attacks. Recall that in order to eclipse a victim, *ping* requests must be sent to the victim repeatedly. A busy traffic flow may be seen as a sign of eclipse attacks. Using these features, we train the data with random forest classification.

Step4: Detection. Our input is the UDP data that has been processed according to a statistical distribution. Our detection model is constructed with *sklearn*, and the test data is split from the collected data in Step1 with a 3:7 ratio. Our node will reboot while it receives many new UDP connections from

adversarial nodes. Our detection can identity an adversary connection request that the eclipse attacks have launched with a high probability.

Table 1: Graph result of RFC detection

	Precision	Recall	F1-score	Support
Normal_visit	0.68	0.98	0.80	158
Attack_visit	0.72	0.93	0.78	127
Avg/Total	**0.71**	**0.95**	**0.62**	**285**

With the random forest classification, the precision rate and recall rate of our detection are 72% and 93%, which are relatively high in practice. The experimental results show that about one third checked attack data can hit to its ground label. Meanwhile, more than 90 percent of all malicious data can be identified correctly, which means most of attack packets can be blocked by our detection model. In the experiment, it costs 1.301s for us to test 2,608 packets on a Thinkpad with Intel Core i5 at 2.5GHz and 8GB RAM.

We note that the eclipse attack was initially defined in the early 2018. There has been few relate work presenting the detection method/framework for the attack. It may be difficult for us compare this work with others in the literature.

5. Related Work

The eclipse attacks on peer-to-peer networks have been studied in these years. [31] presents a survey on the security issues to distributed hash table which is an important building block for peer-to-peer network. A research line of eclipse attack has been discussed in the research works [32, 33] and [34].

The first eclipse attack on a blockchain peer-to-peer protocol, Bitcoin, was presented in [5]. Later, the research work [35] considered to launch eclipse attack by exploiting Ethereum block propagation algorithm. In the attack, an

attacker sends a bogus blockchain to a victim and further exploits a flaw in the block propagation algorithm to prevent the victim from connecting to others. Most recently, the work [1] was agnostic to its block propagation algorithm, and further exploited the attacks to the Ethereum peer-to-peer network. The attack focuses on isolating an Ethereum node from the rest in the blockchain network. The paper discusses two ways to launch eclipse attacks to an Ethereum node - monopolizing connection and table poisoning. Some countermeasures (to the eclipse attacks on Ethereum) are also mentioned in [1]. However, the precondition of these solutions is that system users have to frequently update their Ethereum clients to the latest version. Namely, the solutions strongly depend on the update frequency of users. This may not scale well in practice.

The aforementioned research works have not considered an effective detection mechanism for eclipse attacks. In this paper, we propose an effective model to detect eclipse attacks via data flow, which can safeguard the clients who cannot afford frequently update.

6. Conclusion

To protect Ethereum node from eclipse attacks, we have defined the features of attack connection flow, and further proposed a novel eclipse-attack detection model ETH-EDS. Specifically, we have defined the state change during eclipse-attack process. Our detection model makes use of the fusion context of the model and the stability of fitting degree with traffic increasing to detect malicious connection request. Four types of data packets of UDP are characterized according to the features, namely *packets_size, access_frequencies* and *access_time*. The simulation results show that our model can distinguish the normal traffic from the attack one accurately with a high detection rate and a low false alarm rate.

7. Future Work

Some of our future works are introduced as follows.

Design new features for detection. We believe that there will be more features that can be utilized to train our detection model to identify eclipse attacks. We have investigated two kinds of features, including information entropy and statistical features. In future, we will collect larger scale of data flow and define new features to enhance scalability.

Real-time analysis. Our current model has to collect and store the related data in advance before identifying the classification label of new incoming packets. A real-time detection model based on runtime data flow may provide more practicability.

About using other classifiers. Our detection model has been designed based on the random forest classification. We believe that there must be some other classifiers that may outperform the random forest algorithm. In future, we will consider to leverage various classifiers to make detection comparison w.r.t. accuracy and effectiveness.

We leave the above as the interesting open problems of this work.

Conflict of Interest

None.

Acknowledge

This work is partially supported by the State Key Development Program of China (No. 2017YFE0111900, No. 2017YFB1401201), National Science Foundation of China (No. 61572355, U1736115), Kiban (B) JP18H03240 and JSPS Kiban (C) JP18K11298.

References

[1] Y. Marcus, E. Heilman, S. Goldberg, Low-resource eclipse attacks on ethereum's peer-to-peer network., IACR Cryptology ePrint Archive 2018 (2018) 236.

[2] L. Anderson, R. Holz, A. Ponomarev, P. Rimba, I. Weber, New kids on the block: an analysis of modern blockchains, arXiv preprint arXiv:1606.06530.

[3] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, S. Capkun, On the security and performance of proof of work blockchains, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2016, pp. 3–16.

[4] A. Gervais, H. Ritzdorf, G. O. Karame, S. Capkun, Tampering with the delivery of blocks and transactions in bitcoin, in: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, ACM, 2015, pp. 692–705.

[5] E. Heilman, A. Kendler, A. Zohar, S. Goldberg, Eclipse attacks on bitcoin's peer-to-peer network., in: USENIX Security Symposium, 2015, pp. 129–144.

[6] N. Atzei, M. Bartoletti, T. Cimoli, A survey of attacks on ethereum smart contracts (sok), in: Principles of Security and Trust, Springer, 2017, pp. 164–186.

[7] A. Y. Nur, M. E. Tozal, Record route ip traceback: combating dos attacks and the variants, Computers & Security 72 (2018) 13–25.

[8] H. Wang, H. Guo, M. Lin, J. Yin, Q. He, J. Zhang, A new dependable exchange protocol, Computer communications 29 (15) (2006) 2770–2780.

[9] S. K. Kim, Z. Ma, S. Murali, J. Mason, A. Miller, M. Bailey, Measuring ethereum network peers, in: Proceedings of the Internet Measurement Conference 2018, ACM, 2018, pp. 91–104.

[10] R. Wang, G. Xu, B. Liu, Y. Cao, X. Li, Flow watermarking for antinoise and multistream tracing in anonymous networks, IEEE MultiMedia (4) (2017) 38–47.

[11] A. Kiayias, G. Panagiotakos, On trees, chains and fast transactions in the blockchain., IACR Cryptology ePrint Archive 2016 (2016) 545.

[12] T. Locher, D. Mysicka, S. Schmid, R. Wattenhofer, Poisoning the kad network, in: International Conference on Distributed Computing and Networking, Springer, 2010, pp. 195–206.

[13] G. Xu, J. Liu, Y. Lu, X. Zeng, Y. Zhang, X. Li, A novel efficient maka protocol with desynchronization for anonymous roaming service in global mobility networks, Journal of Network and Computer Applications 107 (2018) 83–92.

[14] S. Chen, M. Xue, L. Fan, S. Hao, L. Xu, H. Zhu, B. Li, Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach, computers & security 73 (2018) 326–344.

[15] G. Xu, Y. Zhang, A. K. Sangaiah, X. Li, A. Castiglione, X. Zheng, Csp-e 2: An abuse-free contract signing protocol with low-storage ttp for energy-efficient electronic transaction ecosystems, Information Sciences 476 (2019).

[16] S. Yu, W. Zhou, W. Jia, S. Guo, Y. Xiang, F. Tang, Discriminating ddos attacks from flash crowds using flow correlation coefficient, IEEE Transactions on Parallel and Distributed Systems 23 (6) (2012) 1073–1080.

[17] B. Wang, Y. Zheng, W. Lou, Y. T. Hou, Ddos attack protection in the era of cloud computing and software-defined networking, Computer Networks 81 (2015) 308–319.

[18] J. Jia, Z. Liu, X. Xiao, B. Liu, K.-C. Chou, psuc-lys: predict lysine succinylation sites in proteins with pseaac and ensemble random forest approach, Journal of theoretical biology 394 (2016) 223–230.

[19] S. T. Zargar, J. Joshi, D. Tipper, A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks, IEEE communications surveys & tutorials 15 (4) (2013) 2046–2069.

[20] T. Karnwal, T. Sivakumar, G. Aghila, A comber approach to protect cloud computing against xml ddos and http ddos attack, in: Electrical, Electron-

ics and Computer Science (SCEECS), 2012 IEEE Students' Conference on, IEEE, 2012, pp. 1–5.

[21] M. R. P. Rad, N. Toomanian, F. Khormali, C. W. Brungard, C. B. Komaki, P. Bogaert, Updating soil survey maps using random forest and conditioned latin hypercube sampling in the loess derived soils of northern iran, Geoderma 232 (2014) 97–106.

[22] M. S. Brieuc, K. Ono, D. P. Drinan, K. A. Naish, Integration of random forest with population-based outlier analyses provides insight on the genomic basis and evolution of run timing in chinook salmon (oncorhynchus tshawytscha), Molecular Ecology 24 (11) (2015) 2729–2746.

[23] E. J. M. Carranza, A. G. Laborte, Random forest predictive modeling of mineral prospectivity with small number of prospects and data with missing values in abra (philippines), Computers & Geosciences 74 (2015) 60–70.

[24] Z. Chen, C. K. Yeo, B. S. L. Francis, C. T. Lau, A mspca based intrusion detection algorithm tor detection of ddos attack, in: Communications in China (ICCC), 2015 IEEE/CIC International Conference on, IEEE, 2015, pp. 1–5.

[25] A. Ghosh, R. Sharma, P. Joshi, Random forest classification of urban landscape using landsat archive and ancillary data: Combining seasonal maps with decision level fusion, Applied Geography 48 (2014) 31–41.

[26] R. Sonobe, H. Tani, X. Wang, N. Kobayashi, H. Shimamura, Random forest classification of crop type using multi-temporal terrasar-x dual-polarimetric data, Remote Sensing Letters 5 (2) (2014) 157–164.

[27] I. Kotenko, A. Ulanov, Agent-based simulation of ddos attacks and defense mechanisms, International Journal of Computing 4 (2) (2014) 113–123.

[28] C. Lindner, P. A. Bromiley, M. C. Ionita, T. F. Cootes, Robust and accurate shape model matching using random forest regression-voting, IEEE

transactions on pattern analysis and machine intelligence 37 (9) (2015) 1862–1874.

[29] ymarcus93, ethpd, https://github.com/ymarcus93/ethpd, accessed August 20 2017.

[30] go ethereum, The encryption algorithm of secp256, https://github.com/ethereum/go-ethereum/blob/master/crypto/secp256k1/secp256.go (27 Aug 2018).

[31] G. Urdaneta, G. Pierre, M. V. Steen, A survey of dht security techniques, ACM Computing Surveys (CSUR) 43 (2) (2011) 8.

[32] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, D. S. Wallach, Secure routing for structured peer-to-peer overlay networks, ACM SIGOPS Operating Systems Review 36 (SI) (2002) 299–314.

[33] E. Sit, R. Morris, Security considerations for peer-to-peer distributed hash tables, in: International Workshop on Peer-to-Peer Systems, Springer, 2002, pp. 261–269.

[34] A. Singh, et al., Eclipse attacks on overlay networks: Threats and defenses, in: In IEEE INFOCOM, Citeseer, 2006.

[35] K. Wüst, A. Gervais, Ethereum eclipse attacks, Tech. rep., ETH Zurich (2016).

Guangquan Xu is a Ph.D. and full professor at the Tianjin Key Laboratory of Advanced Networking (TANK), College of Intelligence and Computing, Tianjin University, China. He received his Ph.D. degree from Tianjin University in March 2008. He is a member of the CCF and IEEE. His research interests include cyber security and trust management.

Bingjiang Guo is a masters student at the College of Intelligence and Computing, Tianjin University, China. He was born in Anyang, Henan Province, China, in 1995. He received a bachelors degree from the Lanzhou University of Technology in June 2017. His current research interests include the blockchain security. He is a Student Member of the IEEE.

Chunhua Su received the BS degree for Beijing Electronic and Science Institute, in 2003 and recieved the MS and PhD degree in computer science from Faculty of Engineering, Kyushu University, in 2006 and 2009, respectively. He is currently working as an assistant professor in School of Information Science, Japan Advanced Institute of Science and Technology. His research areas include algorithm, cryptography, data mining and RFID security & privacy.

Xi Zheng is a PhD in Software Engineering from UT Austin, Master in Computer and Information Science from UNSW, Bachelor in Com-puter Information System from FuDan; now assistant professor/lecturer in Software Engineering at Macquarie University. Specialised in Service Computing, IoT Security and Reliability Analysis. Published more than 40 high quality pub-lications in top journals and conferences. Awarded the best paper in Australian distributed computing and doctoral conference in 2017. Awarded Deakin Re-search outstanding award in 2016. Reviewer for top journals and con-ferences.

Kaitai Liang received the Ph.D. degree from the Department of Computer Science, City University of Hong Kong, in 2014. He is currently an Assistant Professor with the Department of Computer Science, University of Surrey, U.K. His research interests are applied cryptography and information security in particular, encryption, blockchain, post-quantum crypto, privacy enhancing technology, and security in cloud computing.

Duncan S. Wong received the BEng degree from the University of Hong Kong in 1994, the MPhil degree from the Chinese University of Hong Kong in 1998, and the PhD degree from Northeastern University, Boston, MA, in 2002. He is currently working in CryptoBLK, Hong Kong. His primary research interest is cryptography; in particular, cryptographic protocols, encryption and signature schemes, and anonymous systems. He is also interested in other topics in in-

formation security, such as network security, wireless security database security, and security in cloud computing.

Hao Wang is an associate professor in Norwegian University of Science & Technology, Norway. He has a Ph.D. degree and a B.Eng. degree, both in computer science and engineering. His research interests include big data analytics, industrial internet of things, high performance computing, safety-critical systems, and communication security. He has published 80+ papers in reputable international journals and conferences. He served as a TPC co-chair for IEEE DataCom 2015, IEEE CIT 2017, ES 2017 and reviewers for journals such as IEEE TKDE, TII, TBD, TETC, T-IFS, IoTJ, and ACM TOMM. He is a member of IEEE IES Technical Committee on Industrial Informatics. His webpage is *www.haowang.no*.