Computer Security Symposium 2019
21 - 24 October 2019

Proof of data distribution based on trusted hardware

Batnyam Enkhtaivan[1,a)] Pooja Dhomse[1,b)]

Abstract: We consider a scenario where two parties prove to the third party that a communication between them happened. This can be seen in a case of advertisement where the data is distributed by a distributor on behalf of owner to the user. Specifically, we address the possibility of collusion between the distributor and the user in which the owner is deceived to pay for false claim of distribution. In our protocol, the distributor and the user are equipped with trusted hardware. The data is encrypted and decrypted by the trusted hardware of the distributor and the user, respectively, with shared secret key. When decrypting the data, user's hardware generates a proof and sends it to the distributor. This proof is used as the proof of distribution to show to the owner for payment. In such way, we prevent the collusion with the help of the trusted hardware. Moreover, we use the blockchain technology as a method for the ID management and payment.

Keywords: Trusted hardware, Blockchain, Data distribution

1. Introduction

The internet traffic is increasing greatly in the recent years due to more people sharing data - music, images, videos, and etc. - on the internet[1]. And it is expected to increase more as the 5G era arrives with a numerous IoT devices communicating with each other[2], autonomous car connected to the internet, and 8K video streaming[3]. In correspondence with this trend, companies are deploying their services including video streaming on the cloud services such as AWS, Google Cloud, Microsof Azure, and etc. reducing the cost to maintain servers by themselves[4]. Similarly, "information bank" is being considered to be used for collecting, storing and sharing of personal data[5], Roughly saying, the "information bank" is a third party that manages the customers' data similar to the banks who manages their customers' deposit.

In contrast to relying on the centralized cloud and "information bank" services for sharing data, utilization of the distributed network technologies are proposed as another solution to avoid single point of failure and bandwidth costs. IPFS[6] is a peer to peer version controlled file system where each node acts as a server and file is referred by the base-58 encoded hash of the content. Combining IPFS with the blockchain technology[7], Filecoin[8] is a decentralized storage network in which the miners earn the filecoins by providing storage services to clients and thus on the contrary, clients spend the filecoins to miners for storing and distributing the data. Specifiying in video, Videocoin[9] provides a decentralized video encoding, storage, and content distribution network in the form of peer to peer algorithmic market. Videocoin uses proof of retrievability for storage mining, proof of transcoding for compute mining, and a simple messaging protocol for distribution mining. Theta[10] is another network protocol which provides incentive mechanism for decentralized video delivery and streaming by allowing the individual users to share their redundant computing and bandwidth resources. It uses a resource oriented micropayment pool which allows a viewer to do chunk-wise payment for video content pulled from multiple nodes.

Such reliance on the third party shifts the cost of storing and distributing from the customer to the third party. We consider that the cost of distributing includes the usage of the bandwidth which is a limited resource against the increasing traffic as mentioned above. In some aspect, the online advertising industry can be a good example of monetization of the usage of the bandwidth. Advertising platforms earn money from the owners who are willing to advertise their products or services for playing the corresponding advertisements on their distribution platform. This money is paid as per the number of views the advertisement has received[11]. In this scheme, the owners trust the distributors for reporting the correct number of views.

However, there is no trusted way for the owners to verify how many users have actually received the advertisement. In such situations, it is common to use the digital signature of the receiver as a proof that the data is transferred from the distributor to the user[10], [12]. But, it does not prevent the fraud case where the distributor and the user collude to

[1] Security Research Laboratories, NEC Corporation
[a)] b-enkhtaivan@bc.jp.nec.com
[b)] p-dhomse@cj.jp.nec.com

earn money without actually using the distributor's bandwidth for sending data.

In this paper, we present a protocol which addresses both the issues - paid distribution on the network and mutual distrust between the distributor and the owner. We use the blockchain for the identity management and the payment process, and the hardware based Trusted Execution Environment (TEE) to prevent the collusion between the distributor and the user.

The paper is organized as follows. In Sec. 2, we provide general description of the hardware-based TEE. The overview of our protocol is presented in Sec. 3. Then, in Sec. 4, we describe the protocol in details. The experiment results are provided in Sec. 5. Finally, we conclude in Sec. 6

2. Preliminaries: TEE

TEE is an execution environment which is tamper resistant and provide the protected memory and isolated execution space to prevent the regular operating system or applications from controlling or observing the data being stored or processed inside them. Though there are several hardware based trusted execution environments available, we utilize Intel's Software Guard Extensions (SGX)[13], [14] for our implementation because of its wide availability.

Intel SGX is a set of security architecture extensions that enables a Trusted Execution Environment (TEE). The TCB (trusted computing base) of SGX is very small and it only constitutes the CPU's package boundary and software components related to SGX. It allows the creation of secure enclaves that can keep secret. Enclaves are nothing but isolated execution units with encrypted code and data. In addition to this, Intel SGX also has two attestation services - local attestation and remote attestation. Local attestation is a mechanism with which enclave can prove its authenticity to another enclave running on the same platform. Remote attestation is a process through which a client can prove to the service provider that an enclave is running on the SGX compatible machine at the given security level.

In this paper, we use enclave to generate and share the secret encryption key between the trusted hardwares of distributor and user. The trusted hardware on both the ends confirm trustworthiness to each other using remote attestation.

3. Overview

We consider an example of advertising industry where there are three entities - a service provider whom we call as an owner, an advertiser whom we call as a distributor, and a viewer to whom we call as a user. Owner is an individual/organization who owns a particular commercial video of the product or service. Distributor is an individual/organization who is responsible for distributing this commercial to maximum number of users and receive the corresponding distribution fees from the owner. User is an individual who views this commercial and get the knowledge

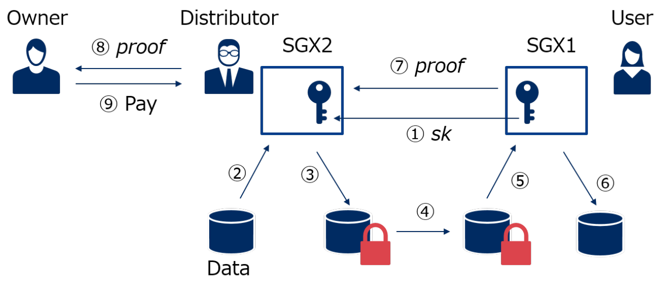


Fig. 1 Setup

about a particular product or service. Owner pays the distributor according to the number of views his commercial has got. But the fundamental problem for the owner is to find the exact number of users who have actually watched or received his commercial. Owner cannot simply trust the distributor as there can be a case of collusion between distributor and user in which they both can cheat owner and obtain the maximum monetary benefit.

We propose a protocol where the user and distributor are equipped with SGX compatible machines SGX1 and SGX2 respectively. SGX1 generates an encryption key and shares with SGX2. SGX2 encrypts the data and send it to SGX1. SGX1 decrypts the data and generates the proof of decryption which is sent to SGX2. SGX2 shares the proof of distribution with the owner. Owner pays the corresponding fees to the distributor.

4. Details

We do not trust the distributor and user for following the protocol. Therefore we use trusted hardware to create digitally signed payloads. Then, if the trusted hardware is not compromised the payload should be correct. We trust the trusted hardware to securely store the secret key. We trust that the symmetric key encryption is appropriately used such as renewing the secret key regularly. We do not consider the recently reported side channel attacks on Intel SGX[15], [16], [17]. We also assume that the ownership of the enclave and the necessary information for attestation and signature verification are registered properly beforehand.

Table 1 Summary of notations

Notation	Description
$dataID$	data ID
i	data chunk index
n	total number of data chunks
Hash()	hash function
$hash_i$	hash of data chunk i
$\langle\cdots\rangle\sigma_{sgx}$	payload signed by SGX
$payload$	output of the SGX2 (distributor's SGX)
$dataChunk_i$	data chunk with index i
$enChunk_i$	encrypted data chunk i
$prevHash$	cumulated hash upto previous data chunk
$proof$	output of the SGX1 (user's SGX)

4.1 TEE-hosted functionalities

The below Fig.1 shows the pseudo code for the TEE

Algorithm 1 TEE-hosted functionalities

```
1: variables
2:     $sk$                ▷ secret key for encryption and decryption
3: end variables
4: function $DivideData$(dataID, $n$)
5:     return $dataChunk_1, dataChunk_2, ..., dataChunk_n$
6: end function
7: function $Encrypt$(dataChunk$_i$)
8:     return $enChunk_i$
9: end function
10: function $Decrypt$(enChunk$_i$)
11:     return $dataChunk_i$
12: end function
13: function $ProcessSGX2$($dataChunk_i$, i, n)
14:     $enChunk_i$=Encrypt(dataChunk$_i$)
15:     $hash_i$=Hash(dataChunk$_i$)
16:     $payload$=$\langle enChunk_i||dataID||i||n||hash_i\rangle\sigma_{SGX2}$
17:     return $payload$
18: end function
19: function $ProcessSGX1$($enChunk_i, prevHash$)
20:     $dataChunk_i = Decrypt(enChunk_i)$
21:     $hash_i = Hash(dataChunk_i)$
22:     $prevHash = (prevHash||hash_i)$
23:     $proof$=$\langle prevHash\rangle\sigma_{SGX1}$
24:     return $dataChunk_i, proof$
25: end function
```

hosted functionalities. Each TEE is equipped with certified keypairs to generate signature. We assume that the verification keys are published to the public bulletin board such as the blockchain. Also, the user's SGX generates a secret key sk for symmetric encryption that is shared with the distributor SGX via secure channel. Such secure channel can be established during the remote attestation between them as described in the manual[13]. We describe each TEE function in the following.

DivideData: divides the data into the number of chunks given to it.

Encrypt: encrypts the input data with the secret key sk using symmetric encryption.

Decrypt: decrypts the input data with the secret key sk using symmetric encryption.

ProcessSGX2: takes in a data chunk, index, the number of division, and returns $payload$ signed with the signing key of the distributor's SGX.

ProcessSGX1: takes in an encrypted data chunk and cumulated hash, and returns the data chunk and the $proof$, which is the cumulated hash signed with the signing key of the user's SGX.

4.2 Process description

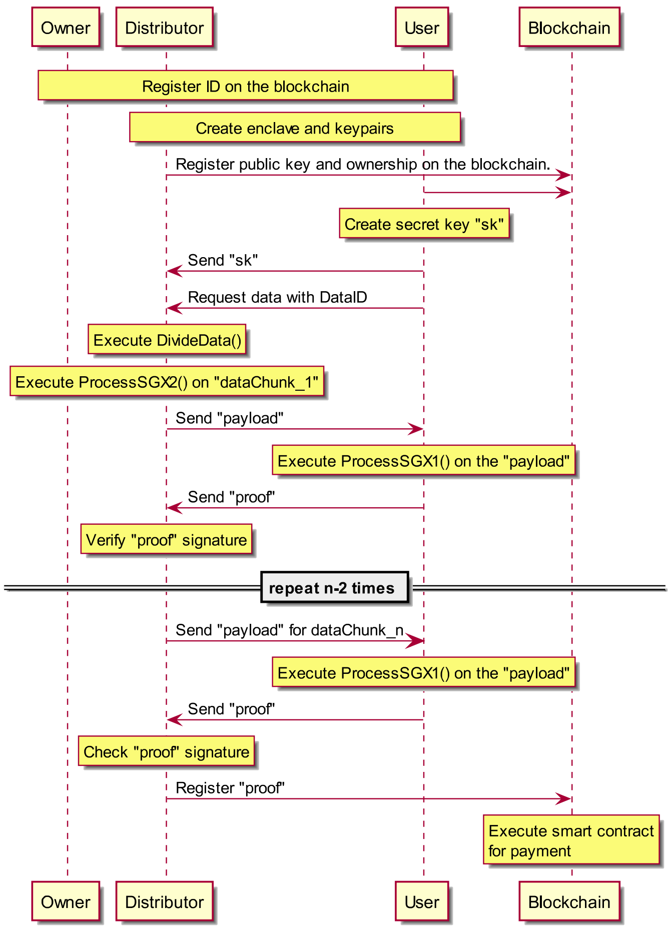
The Fig.2 shows the overall sequence diagram for the protocol. We provide the protocol details stepwise as follows:

Step 1 Owner, distributor, and user register their identifying information on the blockchain. Additionally, distributor and user register the ownership of their respective SGX enclaves and public key on blockchain. Let the user's SGX be SGX1 and distributor's SGX be SGX2. SGX1 generates a secret key sk for symmetric encryption, and then sends the sk to SGX2. The sk is then used for encryption and decryption of the data. Then the user requests for the data corresponding to $dataID$. We assume that the $dataID$ corresponding to the data is publicly available.

Step 2 The distributor divides the data into n data chunks using $DivideData(dataID, n)$ and then executes $ProcessSGX2$ as explained in step 3. There are reasons for deviding this data into chunks. The first reason is to prevent the scenario in which the user receives the data but does not send the $proof$ back to the distributor. In such case, the distributor cannot get paid by the owner. This is similar to the reason provided by Theta[10] protocol. The second reason is due to size limit of the protected memory used for the SGX enclave. It has limit of 128 MB as described in the Intel Software Guard Extensions SDK for Linux OS[18].

Step 3 The distributor executes $ProcessSGX2(dataChunk_1, 1, n)$ to obtain $payload$ for $dataChunk_1$. Then, the distributor sends the $payload$,

$$\langle enChunk_1||dataID||1||n||hash_1\rangle\sigma_{SGX2}, \quad (1)$$



Owner
Distributor
User
Blockchain
Register ID on the blockchain
Create enclave and keypairs
Register public key and ownership on the blockchain.
Create secret key "sk"
Send "sk"
Request data with DataID
Execute DivideData()
Execute ProcessSGX2() on "dataChunk_1"
Send "payload"
Execute ProcessSGX1() on the "payload"
Send "proof"
Verify "proof" signature
repeat n-2 times
Send "payload" for dataChunk_n
Execute ProcessSGX1() on the "payload"
Send "proof"
Check "proof" signature
Register "proof"
Execute smart contract for payment

Fig. 2 Sequence diagram

to the user.

Step 4 The user then executes $ProcessSGX1$ in which the SGX1 first decrypts $enChunk_i$ and retrieves $dataChunk_i$. Then the SGX1 calculates the hash for this $dataChunk_i$. This hash is combined with the $prevHash$ and stored as new $prevHash$. Then, the user obtains the $proof$ as signed cumulated hash $prevHash$. and the $dataChunk_i$. The user sends the $proof$ to the distributor to receive the next $payload$ for $dataChunk_2$.

Step 5 The distributor verifies the signature of the cumulated hash value. Also, the distributor checks if the value of the cumulated hash is correct. If both the conditions are satisifed, the distributor sends the payload with the $dataChunk_2$.

Step6 After repeating the step 3 - 6 for for all the data chunks, the distributor has the last $proof$.

Step7 The distributor register the last $proof$ on blockchain and the smart contract for payment is executed.

4.3 Role of blockchain

In above mentioned protocol, we propose to use blockchain for below three purposes. Permissioned blockchain or permissionless blockchain, any of them can be used. It depends on the application requirements in which the protocol will be used.

- **ID management**: it can be done by recording the registration IDs of owner, distributor and user. Also, distributor and user record their indivisual public keys and enclave identifiers on blockchain
- **Proof logging**: the distributor can submit the final proof of distribution on blockchain. This can be verified by the owner before making any payment to the distributor. Since, the number of such proofs in case of large number of users will be more, the owner might not be able to check all the proofs. In such case, a trusted third party can be allocated to do this job. Alternatively, the owner can opt to randomly sample the proofs and verify only those proofs
- **Payment process**: either the owner verifies the proof and pays, or a smart contract for payment be executed to do the corresponding payment to the distributor is possible.

5. Experiment

We implement this protocol on SGX compatible machine for the different values of data size. We used Intel(R) Xeon(R) CPU E3-1270 v6 @ 3.80GHz, 32 GB RAM. The OS is Ubuntu 16.04.4. As to the cryptographic protocols, we utilize AES-GCM 128 bit for symmetric encryption, SHA256 for hashing umplemented in the SDK of the Intel's SGX. We have measured execution time for these operations. The Fig.3 shows relation between the execution time and the chunk sizes.

In a general observation, it seems unusual that hashing time is larger than encryption and decryption time. However, we find that similar result is presented by D. Harnik[19] comparing different implementations of these functions on the Intel's SGX. Such reversing of the execution time of the encryption and hashing seems to be due to the implementation of the SGX SDK.

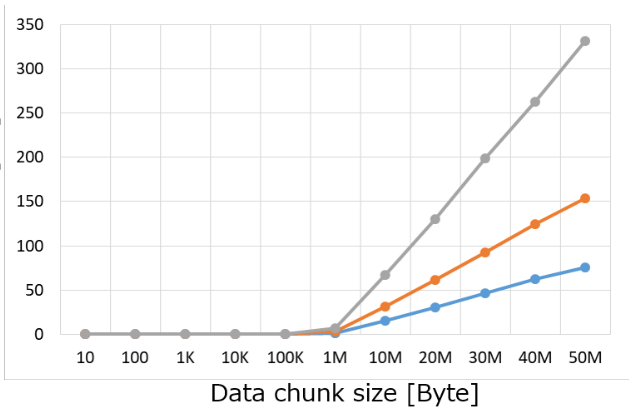


Fig. 3 The measured time for encryption, decryption (AES-GCM 128 bit) and hashing (SHA256) against varying data chunk sizes.

The execution time increases linearly as the data chunks size increases. We find that even for 50 MB size chunks the execution times are in the order of 100 ms. Therefore, to transfer 1 TB file (2000 data chunks), our protocol takes few minutes. This is similar to the network transfer time for 10 GB ethernet network if we apply simple calculation. So, we think our protocol is feasible in the real application.

We find that if the data chunk size goes beyond 50 MB, the program can be executed, but the output is incorrect. This happens because the data size is larger than the enclave memory size. The allowed memory size for SGX is 128 MB. However, the actual memory size seems to be around 100 MB as this blog states[20].

So, if we allocate several enclaves, the memory size for each enclave would be smaller. However, we find that this size limit can be increased on a system that supports paging such as linux as discussed in this forum[21].

6. Conclusion

It is important that data transfer on the network should be charged as it consumes the network bandwidth. However, in scenarios where the distribution of the data is done by a third party on behalf of the owner - a case of advertising domain, it is necessary to prove that the user has actually received the commercial. To accomplish this, we have utilized digital signature and trusted hardware technology of Intel SGX to generate the proof of data distribution. Due to this, the trust is moved from human to hardware. If there are auditing requirements in a particular application, the same protocol can be enhanced by using blockchain technology for logging the several operations which can later be audited by the auditor.

References

[1] Cisco Visual Networking Index: Forecast and Trends, 2017-2020, available from ⟨https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.pdf⟩

[2] available from ⟨https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016⟩

[3] available from ⟨https://www.gartner.com/en/newsroom/press-releases/2018-12-18-gartner-survey-reveals-two-thirds-of-organizations-in⟩

[4] available from ⟨https://www.gartner.com/en/newsroom/press-releases/2019-04-02-gartner-forecasts-worldwide-public-cloud-revenue-to-g⟩

[5] available from ⟨https://www.dac.co.jp/english/press/2018/20180910_ib⟩

[6] IPFS is the Distributed Web, available from ⟨https://github.com/ipfs/ipfs⟩ (2019.07.29).

[7] Satoshi, N.: Bitcoin: A Peer-to-Peer Electronic Cash System, available from ⟨https://bitcoin.org/bitcoin.pdf⟩ (2008).

[8] Protocol Labs.: Filecoin; A Decentralized Storage Network, available from ⟨https://filecoin.io/filecoin.pdf⟩ (2017).

[9] Devadutta, G.: VideoCoin - A Decentralized Video Encoding,Storage, and Content Distribution Network, available from ⟨https://storage.googleapis.com/videocoin-preico/VideoCoin-Whitepaper.pdf⟩ (2017).

[10] Theta: A Decentralized Video Delivery and Streaming NetworkPowered by a New Blockchain, available from ⟨https://s3.us-east-2.amazonaws.com/assets.thetatoken.org/Theta-white-paper-latest.pdf?v=1564378087.136⟩ (2018).

[11] Ebtessam, E; The effectiveness of the Pay Per Click PPC model in today's business, available from ⟨https://www.academia.edu/29728426/⟩ (2015).

[12] Król, M; Sonnino, A; Al-Bassam, M; Tasiopoulos, A; Psaras, I; Proof-of-Prestige: A Useful Work Reward System for Unverifiable Tasks, IEEE International Conference on Blockchain and Cryptocurrency, (2019).

[13] Intel Corporation; Intel 64 and IA-32 Architectures Software Developer's Manual, Sep 2015. Reference no. 325462-056US.

[14] Costan, V; Devadas, S; Intel SGX Explained, Published in IACR Cryptology ePrint Archive 2016, available from ⟨https://eprint.iacr.org/2016/086.pdf⟩.

[15] Lipp, M; Schwarz, M; Gruss, D; Prescher, T; Haas, W; Fogh, A; Horn, J; Mangard, S; Kocher, P; Genkin, D; Yarom, Y; Hamburg, M; Meltdown: Reading Kernel Memory from User Space, 27th USENIX Security Symposium, (2018).

[16] Bulck, J. V; Minkin, M; Weisse, O; Genkin, D; Kasikci, B; Piessens, F; Silberstein, M; Wenisch, T. F; Yarom, Y; Strackx, R; FORESHADOW: Extracting the Keys to the Intel SGX Kingdom withTransient Out-of-Order Execution, 27th USENIX Security Symposium, (2018).

[17] Kocher, P; Horn, J; Fogh, A; Genkin, D; Haas, W; Hamburg, M; Lipp, M; Mangard, S; Prescher, T; Schwarz, M; Yarom, Y; Spectre Attacks: Exploiting Speculative Execution, available from ⟨https://meltdownattack.com/⟩ (2018)

[18] Intel Software Guard Extensions SDK for Linux OS, available from ⟨https://01.org/sites/default/files/documentation/ intel_sgx_sdk_developer_reference_for_linux_os_pdf.pdf⟩ (2016).

[19] Harnik, D; Impressions of Intel SGX performance, available from ⟨https://medium.com/danny_harnik/impressions-of-intel-sgx-performance-22442093595a⟩ (2017).

[20] Machida, T; Yamamoto, D; Morikawa, I; A Survey and Analysis on Intel SGX and Its Demonstrations, SCIS, (2017).

[21] available from ⟨https://software.intel.com/en-us/forums/intel-software-guard-extensions-intel-sgx/topic/670322⟩ (2016).