Journal of Advances in Science and Technology
Vol. 15, Issue No. 1, March-2018, ISSN 2230-9659

Data Provenance for Secure Internet of Things

Dr. Sridevi*

Department of Computer Science, Karnatak University, Dharwad

Abstract – Data provenance to maintain data integrity and authenticity is a significant challenge in the Internet of Things (IoT) environments. Additionally, if the provenance metadata itself can be communicated in a privacy preserving manner, it expands the usage of IoT systems to human societal domains where privacy is of paramount importance. These papers present a scheme to combine data provenance and privacy-preserving solutions. Proposed scheme merges Physical Unclonable Function (PUF) technology with non-interactive zero-knowledge proof to provide trustworthy and dependable IoT systems. In this context, the IoT device can anonymously send data to the corresponding server associated with the proof of ownership. Proposed a privacy-preserving data provenance protocol. This protocol was synthesized with Altera Quartus. It was implemented on an Altera Cyclone IV FPGA to demonstrate its practicality and feasibility.

Keywords: IoT, AES, Diffie Hellman, etc.

----------X----------

1. INTRODUCTION

The Internet of Things (IoT) technology deployment has been growing exponentially within the last decade. IoT's are everywhere from a smart and connected home, to hospitals, to military and agriculture. This is still the proverbial tip of an iceberg. The ceiling for IoT deployment still has much further to go. This growth brings along several challenges, especially in the area of cyber-security. Provenance and privacy preservation are considered two important factors within IoT cyber-security domain due to the fact that the data is transmitted over communication channels. More specifically, in an IoT system, data provenance refers to the metadata that describes the ownership, creation process, and modification of data. Providing secure data provenance aims to establish the trust in the data collected among the IoT devices. Moreover, since IoT networks are ideally open systems to allow plug-in functionality extension, the data provenance should be communicated in a way so that the privacy of the provenance provider is not violated by leaking unnecessary information. This is what a privacy preserving data provenance model seeks to establish. Physical Unclonable Functions (PUF) is good candidates for providing a unique device-specific identity. Such unique silicon biometric identities can be a good source of data provenance. Software PUF (SW-PUF) composes the silicon fabrication process variation with the software input denied execution paths to generate reproducible randomness that is both device and software dependent to serve as a hardware-software fingerprint. This functionality allows the SW-PUF to provide unique metadata to certify if a specific IoT device executed a specific data creation or modification program

Proposed a novel privacy preserving data provenance model based on Physical Unclonable Functions and Non-Interactive Zero-Knowledge Proof systems. This framework guarantees that the received data from an IoT device is collected from a registered authorized device; that it can be verified that the said authorized device ran a specific authorized data creation or modification program; and that the preceding two properties can be established without revealing the device identity. Specifically, the proposed solution contributes to achieving the following security goals:

- Source Identity Authenticity: guarantees that the data originated from the specific IoT device that sent it.
- Privacy-Preserving Identity: ensures that the real identity of the owner of the data is not unveiled.
- Data Integrity: confirms that the data transmitted is not tampered with.
- Device Trust: ensures that the device is not exploited by a malicious code.

2. PRIVACY PRESERVING DATA PROVENANCE PROTOCOL

The proposed protocol encompasses four stages. The first stage (called the Setup and Enrolment stage) is to generate public parameters required by

the following stages in the protocol. These parameters are associated with an IoT device profile indexed by a virtual ID assigned to an IoT device by the server. The second stage, called authentication stage, is to prove the identity of an IoT device to the server ensuring its privacy. The third stage, called Key Exchange stage, is to exchange a symmetric key between a server and an IoT device. The last stage, called Data Transmission and Verification stage, is to start trusted communication between an IoT device and the server which confirms the source of the transmitted data. Proposed design is based on software PUF and non-interactive zero-knowledge proofs. The SW-PUF is used to provide a proof of identity and root of trust for an IoT device. This type of PUF ensures that the data generated and processed by an IoT device is measured as it is computed inside the IoT device itself. This is beneficial in proving the provenance of the data. Non-interactive zero-knowledge proofs were used to ensure secure privacy-preserving communication between an IoT device and a server in the authentication process. The elliptic curve cryptography over Binary Fields GF(2^m) was chosen in order to reduce the computational requirements for the IoT devices while maintaining the security level of other mathematical frameworks.

3. ENROLMENT AND SETUP STAGE

This stage is performed only once when an IoT device is deployed in the field for the first time. An IoT device and a server prepare all parameters required to perform the authentication protocol in the future and agree on a virtual ID for communication with the IoT device. The following steps will be performed by an IoT device to generate public parameters required by the following stages:

Note, none of the parameters generated in this stage contain sensitive information, thus, it is safe to transmit them over an open communication channel.

Device:

- Select elliptic curve E over Binary Fields GF (2^m)
- Choose base point G = (Gx; Gy). Not that in ECC, many parameters like G are points in 2-dimensional space. We will often refer to the x and y components of such points by notation Gx and Gy respectively.
- Compute public Key A = x_G where x is the SW-PUF signature that has been generated during the bootup of the device.
- Share the public parameters fG = (Gx; Gy);A = (Ax; Ay)g with the server.

Server:

- Generate Virtual id (V id) for the device to use in future communications.
- Store the public parameters associated with this V id.

Authentication stage

Proposed protocol uses a unique SW-PUF signature (x) that can be generated during the bootup phase of the device to authenticate the execution environment of the device, or it can be generated every time a new data is produced or processed to authenticate the data creation or modification step at a specific IoT device. If a log of a sequence of data creation and modification events at a specific IoT device needs to be authenticated, then a chained hash of these raw SW-PUF signatures in the order of events' occurrence is needed. This is similar to the way a TPM maintains platform configuration registers (PCRs). Then, this signature hash needs to be saved in a protected memory in the IoT device that we will refer to as Metadata Tracking Register (MTR). MTR's role is similar to the TPM's PCR. It holds a chained hash of provenance metadata evolution through creation and modification steps.

The MTR can only be updated through MTR extension API which takes the hash of current MTR value concatenated with the new SW-PUF signature *val* as the new MTR value

MTR←h(MTR(v)||val), same as the PCR extension. Figure 1 shows the proposed protocol for this stage. The authentication protocol is based on non-interactive zero-knowledge proofs These protocols can prove the knowledge or possession of a value to a verifier without requiring multiple interactive steps (as in traditional zero-knowledge proofs). This verification leaks zero information about the value known to the prover. Let the value known to the prover be x. The prover generates two derived values from x, a t-value given by t = f_t (x;K) - a function of the secret value x and several public parameters such as a key K, and potentially others such as nonces; a s-value given by s = f_s(x;K) - a function of the secret value x and several public parameters such as a key K, and potentially others such as nonces. The functions f_s and f_t allow the verifier to check on some mathematical properties of s and t combined which is not likely to hold unless the prover knows x. But s and t together do not reveal x. The IoT device (Prover) performs the following steps every time the device boots up to start a trusted communication with the server (Verifier):

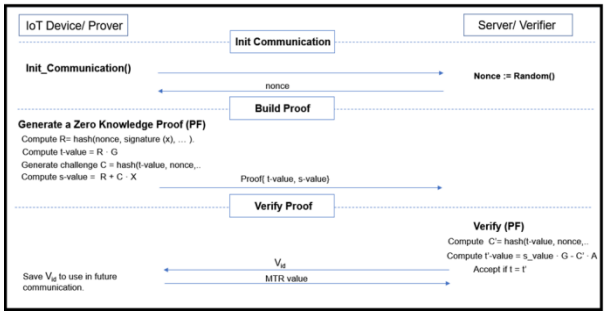

Figure 1: Authentication protocol

Prover: Ask the Server (Verifier) to initiate the communication.

Verifier: Send a random nonce to the prover to ensure that old communications cannot be reused.

Prover:

- Compute v=hash (nonce||x||all public parameters…..).
- Compute t-value where t=v.G Note: G is public parameter generated during enrolment
- Generate challenge c=hash(t-value||all public parameters||…)
- Compute s-value where s=v+c.x
- Send t-value, s-value to the verifier.

Verifier:

- Compute c'= hash(t-value||all public parameters||…)
- Compute t'-value=s.G- c'. Note that A is public key generated and published during enrolment.
- Verify that t-value= t'-value holds.
- Reject in case of mismatch

Once the server (verifier) authenticates the IoT device (prover) s/he sends Virtual id (V id) to

the device for future communications. Note that V id was generated during enrollment, but not shared with the IoT device until after authentication. The IoT device could update the metadata tracking register (MTR) value as follows: $MTR_{new} \leftarrow hash(MTR_{old}$ = SW - PUF at bootup).

This version of MTR however reveals the raw SW-PUF bootup signature. We need to hide it

with some other random parameters. However, cannot use non-reproducible parameters such as time or a random nonce. An MTR digest should be reproducible for the same sequence of bootup, data creation, and data modification events for the verifiability, just as PCR digests are. One solution to this is to create a special program module called nonce-parameter-generator (seed). When this program executes, SW-PUF generates its signature, which can be used as a nonce like hiding parameter. The seed could be the bootup SW-PUF signature in MTR_{old}, which should be reproducible for the future device boot ups. This is the version we propose to use:

$MTR_{new} \leftarrow$ hash(MTR_{old}||nonce - parameter - generator(MTR_{old})). The prover/device sends

this MTR_{new} to the server/verifier to be used in the future steps.

4. KEY EXCHANGE STAGE

To ensure secure communication between the IoT device and the server, we use symmetric

AES encryption algorithm for better efficiency in place of an asymmetric encryption system. Use a hash function with a standard key exchange based on elliptic curve Diffie-Hellman protocol to generate and exchange the AES key between the two parties. The following explanation outlines the main steps:

Device:

- Choose private key k_d where $k_d < 2^m$.
- Compute public Key $Q_d = k_d .G$. Recall that m and G are public parameters.
- Share the public parameters (Q_dx;Q_dy) with the server.

Server:

- Choose private key k_s where $k_s < 2^m$.
- Compute public Key $Q_s = k_s .G$
- Share the public parameters (Q_sx;Q_sy) with the IoT device.

Device:

- Compute shared Key $K_dS = k_d , Q_s$
- Compute AES key $K_{AES} = hash(K_{ds})$

Server:

- Compute shared Key $K_{ds} = k_s .Q_d$
- Compute AES key $K_{AES} = hash(K_{ds})$

5. DATA TRANSMISSION AND VERIFICATION STAGE

Proposed a transmission and verification protocol for securing the data transmission and providing verification of the provenance in IoT environments. AES algorithm has been chosen to encrypt/decrypt all the transmitted messages. The proposed protocol consists of three phases: Init Communication: where the IoT device initiates the communication by sending the V id and the stored MTR value. Generating Data: where the IoT device generates both the data and the metadata and sends them the server. Verify Data Provenance: where the server verifies the data by confirming the source of the data. The proposed protocol is shown in Figure 2. The following steps describe each phase. This protocol needs to be repeated for every data transmitted from an IoT device to a server:

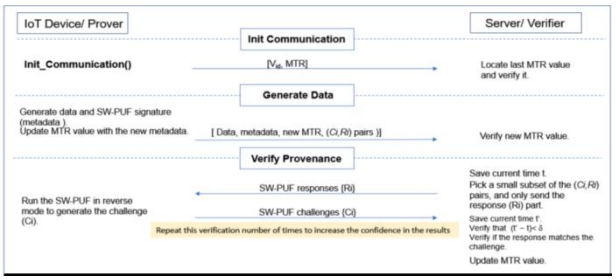

Figure 2: Data Transmission and Verification protocol

Device:

- Ask the Server (Verifier) to initiate the communication.
- Send Vid and current MTR value.

Server:

- Locate the MTR value for the received V id and verify it.
- Reject in case of mismatch.

Device:

- Generate the data and SW-PUF signature (metadata) of a data creation/modification event generated by the SW-PUF.
- Update MTR value with the new metadata.
- $MTR_{new} \leftarrow hash(MTR_{old}||SW\ PUF\ signature)$.
- Send Data and metadata consisting of MTR_{new} and SW-PUF (Ci;Ri) pairs to the server.

Server:

- Verify the received metadata MTR_{new} and SW PUF signature against the old value
- MTR_{old} associated with V id by checking the equality $MTR_{new} = hash(MTR_{old}||SW\ PUF\ signature)$.
- Reject in case of mismatch.
- Save current time t.
- Pick a small subset of the helper SW-PUF (Ci;Ri) pairs. Verify the existence and integrity of SW-PUF by sending the response Ri part of the selected subset asking the prover to generate the corresponding Ci part through reversible computation.

Device:

- Run the SW-PUF in reverse mode to generate the challenge Ci corresponding to the received Ri.
- Send Ci to the server.

Server:

- Save current time t'.
- Verify that (t'- t) < **δ.** This check ensures that Ci is computed in a reverse mode. An untruthful device would have needed more than **δ** time to perform additional computations or to retrieve it from a secondary storage. Note that we also assume that relative to the IoT device small cache, the (Ci;Ri) sets are significantly larger - preventing a cached response to bypass reverse computation.
- Verify if the received response $Ri_{received}$ matches the response Ri from the (Ci;Ri) pairs
- received at the MTR verification stage.
- Reject in case of mismatch.
- Update MTR value.

6. THREAT MODEL

Following are some of the objectives of an attacker for the proposed protocol:

- Mimic an IoT device and transfer maliciously modified data to the server. This attack cannot be applicable in our scheme since the IoT device identity is based on PUF which makes it close to impossible to generate or clone a fake identity.

- Tamper or modify the data sent by a valid IoT device. This attack can be detected by the proposed verification protocol, where the MTR value will not match the saved MTR value at the server.

6.1 Implementation

Complete design of the proposed privacy-preserving data provenance protocol has been modelled in Verilog (HDL), simulated by ModelSim XE, synthesis with Altera Quartus and implemented on FPGA Altera Cyclone iv at speed of 50 MHz. The AES-128 encryption and decryption algorithm, SHA-256 hash function, ECC over Binary Fields GF(2^{233}) engine. All the other proposed protocols in this paper also map to the FPGA. The SW-PUF is the only component not mapped to the FPGA since reversible computation using transmission gate logic is not feasible in an FPGA fabric. The data and the metadata (SW-PUF signatures) generation was performed in software using HSPICE k-2015.06 and Pin tool (dynamic binary instrumentation tool).

Public-key cryptography used in the enrolment and authentication stages was based on an Elliptic curve over the binary field. The ECC is suitable for resource constrained system because it can offer the same security level as other asymmetric systems for a much smaller key size. This implementation used the recommended Curve B-233 presented in the NIST FIPS Locke and Gallagher (2017) to provide excellent security level. The 233 bits key size has performance comparable to RSA 2048 bits key size.

6.2 Results

Implementation consumes around 40K Logic Elements (LEs) for the IoT device and around 37K LEs for the IoT server. Table 1 reports the LEs needed for each step in our protocol. It takes around 118µ sec to perform the enrolment, authentication, and key exchange protocols. About 120m sec is required to transfer and to verify the provenance of 1 megabyte of data and metadata. Table 2 records the average execution time for each step in our protocol. Note that all the verification steps take about the same time, but the transmission & verification time dominates.

Table 1 Performance results (Total Logic Elements(Les))

Protocol	IoT Server	IoT Device
Enrolment	-	109
Authentication	1,693	729
Key Exchange	22,941	22,941
Transmission and verification	16,670	14,170

Table 2 Performance Results (Execution Time)

Protocol	IoT Server	IoT Device
Enrolment	-	34u sec
Authentication	40.1u sec	40.0u sec
Key Exchange	42.5u sec	42.5u sec
Transmission and verification	120.2m sec	71,6m sec

7. CONCLUSION

In this paper, privacy-preserving data provenance solution that merges Physical Unclonable Function (PUF) technology with non-interactive zero knowledge proof to provide trustworthy and dependable IoT systems. Proposed scheme, an IoT device can anonymously send data to an IoT server. The server enrols an IoT device and verifies all the provenance metadata for data creation and modification. The proposed protocol has been designed and synthesized with Altera Quartus and implemented on FPGA Altera Cyclone iv.

REFERENCES

[1] Abdel-Basset, M., Manogaran, G., Mohamed, M., and Rushdy, E. (2017). Internet of things in smart education environment: Supportive framework in the decision-making process. Concurrency and Computation: Practice and Experience, 31(10):e4515.

[2] Adhikary, T., Jana, A. D., Chakrabarty, A., and Jana, S. K. (2017). The internet of things (iot) augmentation in healthcare: An application analytics. In International Conference on Intelligent Computing and Communication Technologies, pages 576-583. Springer.

[3] Alharbi, K. and Lin, X. (2017). Pdp: A privacy-preserving data provenance scheme. In 2017 32nd International Conference on Distributed Computing Systems Workshops, pages 500-505. IEEE.

[4] Aman, M. N., Chua, K. C., and Sikdar, B. (2017). Secure data provenance for the internet of things. In Proceedings of the 3rd ACM International Workshop on IoT Privacy, Trust, and Security, pages 11-14. ACM.

[5] Conti, M., Dehghantanha, A., Franke, K., and Watson, S. (2018). Internet of things security and forensics: Challenges and opportunities.

[6] Jaigirdar, F. T., Rudolph, C., and Bain, C. (2017). Can i trust the data i see?: A

physician's concern on medical data in iot health architectures. In Proceedings of the Australasian Computer Science Week Multiconference, page 27. ACM.

[7] Javaid, U., Aman, M. N., and Sikdar, B. (2018). Blockpro: Blockchain based data provenance and integrity for secure iot environments. In Proceedings of the 1st Workshop on Blockchain-enabled Networked Sensor Systems, pages 13-18. ACM.

[8] Kamal, M. et al. (2018). Light-weight security and data provenance for multi-hop internet of things. IEEE Access, 6: pp. 34439-34448.

[9] Lu, Y. and Da Xu, L. (2018). Internet of things (iot) cybersecurity research: a review of current research topics. IEEE Internet of Things Journal, 6(2): pp. 2103-2115.

[10] Ray, P. P. (2018). A survey on internet of things architectures. Journal of King Saud University- Computer and Information Sciences, 30(3):pp. 291-319.

Corresponding Author

Dr. Sridevi*

Department of Computer Science, Karnatak University, Dharwad