Key Establishment Using Physically Unclonable Functions

SeongHan SHIN and Kazukuni KOBARA
National Institute of Advanced Industrial Science and Technology (AIST)
2-4-7 Aomi, Koto-ku, Tokyo, 135–0064, Japan
Email: seonghan.shin@aist.go.jp

Abstract—As an alternative to tamper-resistant modules, PUFs (Physically Unclonable Functions) [1], [6], [7], [8], [9], [10], [11], [14], [15], [18], [25], [26] have received much attention due to their unclonable physical characteristics. In this paper, we propose key establishment (KE1 and KE2) protocols using PUFs which can mitigate side-channel attacks and satisfy two security requirements (i.e., Freshness and Non-Regeneration). Also, we discuss security of the KE1 and KE2 protocols, and compare with the previous PUF-based key exchange protocols [3], [4], [27].

1. Introduction

In recent years, the needs of cryptographically secure keys have been rapidly increasing for various field applications such as IoT (Internet of Things), M2M (Machine to Machine) communication and CPS (Cyber Physical Systems). Some applications actually require cryptographic keys in order to protect their communication data from eavesdropping, modifications, impersonations, and so on. Ideally, cryptographic keys should be stored in perfect tamper-resistant modules, but realizing perfect tamper-resistance with low costs is still a challenging open problem. Therefore, better solutions also providing security against side-channel attacks (e.g., differential power analysis [13]) are desirable. As an alternative to tamper-resistant modules, PUFs (Physically Unclonable Functions) [1], [6], [7], [8], [9], [10], [11], [14], [15], [18], [25], [26] have received much attention due to their unclonable physical characteristics (e.g., micro- or nanoscale structural disorder and minuscule manufacturing irregularities).

In this paper, we propose key establishment protocols using PUFs which can mitigate side-channel attacks and satisfy two security requirements (i.e., Freshness and Non-Regeneration). Please, see the following sections for more details.

2. Assumptions and Requirements

In this section, we explain assumptions and security requirements for key establishment protocols using PUFs.

Assumption 1: It is hard for an attacker to let the PUF generate exactly the same response or responses whose Hamming distances are closer than the exhaustively searchable area for the same challenge.

Assumption 2: Modules are equipped with low-level tamper-resistance and an attacker cannot read a memory in the modules directly. But, side-channel information leaks out from an internal value that depends on a part of secrets whose space is exhaustively searchable, and that can be changed with known values by the attacker.

Assumption 1 holds by making length of the response larger (e.g., [11]) and/or combining multiple responses. And Assumption 2 is true at least against differential power analysis [13]. We stress that these assumptions are reasonable because the maximal amount of randomness/entropy in a physical system is polynomially bounded in the size of the system [2], and several attacks on PUFs themselves have been reported recently (e.g., [22], [23]).

Requirement 1 (Freshness): New and independent common keys should be shared between two modules after successful key establishment.

Requirement 2 (Non-Regeneration): It should be hard for an attacker to regenerate the same secrets in the modules where a certain internal value (which is possibly used for side-channel attacks) depends on a part of the regenerated secrets whose space is exhaustively searchable, and that can be changed with known values by the attacker.

Freshness is required to refresh session keys (possibly to be revealed by side-channel attacks), and Non-Regeneration is needed to make side-channel attacks (especially, differential power analysis) much harder.

3. Bad Examples

Before showing our proposals, we give some bad examples so as to get useful lessons in regard to Freshness and Non-Regeneration.

3.1. First Example (Pre-Shared Key in Memory)

In this example, two modules share the same secret beforehand, and then generate fresh session keys depending on exchanged nonce values and the pre-shared secret.
This allows to regenerate the same pre-shared key in the module where an internal value depends on a part of the pre-shared secret, and the internal value is changed with the nonce values. Hence, an attacker can collect a lot of side-channel information on the internal value and the (guessed) pre-shared secret, and then can apply powerful side-channel attacks. The same method holds even if a private (i.e., decryption/signature) key for a public-key cryptosystem is used in order to establish fresh session keys.

3.2. Second Example (Key Update)

In this example, two modules update the key stored in the module after successful key establishment.

Though this loads different secrets in the memory every time after successful key establishment, an attacker can halt it so that the same secret would be used repeatedly.

3.3. Third Example (Typical Use of PUF for Key Generation [12], [24])

This example proceeds as follows:

1) To let a module hold a cryptographic key, Center sets a helper data \( D = R \cdot H \) to the module where \( H \) is an \( n \times (n-k) \) parity check matrix of the underlying error correction code (or a CRC (Cyclic Redundancy Check) generator) if the underlying error correction code is cyclic, \( R \) is a binary vector of \( n \) coordinates (which is a response of the PUF to a challenge \( C \)), and \( ' \cdot ' \) is an inner product.

2) In order to restore the key, the module accepts the same challenge \( C \) and then obtains its response \( R' \) from its PUF. Since \( R' \) is usually a little bit different from \( R \), it applies a syndrome decoding to \( R' \cdot H + D \) and obtains \( E = R' + R \) where \( '+' \) is an exclusive-or operation. Finally, it calculates a keying material \( KM \) with \( KM = R' + E = R \).

3) The keying material \( KM \) is then used to generate a session key after hashing it or applying a key derivation function [5] to it.

The above satisfies neither Freshness nor Non-Regeneration because the purpose of this example [12], [24] is to restore the same key every time in the memory using PUF. The restored key may be used to establish fresh session keys between modules, but it does not satisfy Non-Regeneration.

3.4. Fourth Example (Naive Key Establishment Using PUF)

This example, which establishes fresh session keys using PUF, proceeds as follows:

1) In order to let Module1 and Module2 establish the same (but changing every time) session key, Center sets a helper data \( D_1 = R_2 + N \) to Module1 and \( D_2 = R_1 + N \) to Module2, respectively, where \( R_1 \) and \( R_2 \) are responses of the PUFs in the modules to the respective challenges \( C_1 \) and \( C_2 \), and \( N \) is a random code word of the PUFs in the modules.

2) To establish a fresh session key with Module2, Module1 accepts the challenge \( C_1 \), obtains its response \( R'_1 \) from its PUF, and then sends the syndrome of \( R'_1 \) (i.e., \( S'_1 = \tilde{R}'_1 \cdot H \)), to Module2. In the same way, Module2 accepts the challenge \( C_2 \), obtains its response \( R'_2 \) from its PUF, and then sends \( S'_2 = R'_2 \cdot H \) to Module1.

3) Module1 applies the syndrome decoding algorithm to \( S'_2 + D_1 \cdot H \) and then obtains \( E_2 = R_2 + R'_2 \). It calculates a keying material \( KM \) with \( KM = D_1 + E_2 + R'_1 \), which is equivalent to \( N + R_2 + R'_1 \).

In the same way, Module2 applies the syndrome decoding algorithm to \( S'_1 + D_2 \cdot H \) and then obtains \( E_1 = R_1 + R'_1 \). It calculates a keying material \( KM \) with \( KM = D_2 + E_1 + R'_2 \), which is equivalent to \( N + R_1 + R'_2 \).

4) The keying material \( KM \) is then used to generate a session key after hashing it or applying a key derivation function [5] to it.

Even though this example enables both modules to establish the fresh session key every time (for the same pair of challenges \( C_1 \) and \( C_2 \)), it does not satisfy Non-Regeneration since an attacker can regenerate previously-established session keys in the module. See the next subsection.

3.5. Key Regeneration Attack on Fourth Example

A key regeneration attack on the fourth example is as follows:

1) An attacker eavesdrops communications between Module1 and Module2, and then records a pair of exchanged \( S'_1 \) and \( S'_2 \).

2) The attacker intrudes in the middle of Module1 and Module2, intercepts new \( S''_1 \) and \( S''_2 \), and then sends \( S''_1 = S'_1 + S''_2 + S'_2 \) to Module2 instead of \( S'_1 \).

3) After receiving \( S''_1 \), Module2 applies the syndrome decoding algorithm to \( S''_1 + D_2 \cdot H \) by following the procedure. Since \( S''_1 + D_2 \cdot H = S'_1 + S''_2 + S'_2 + D_2 \cdot H \), it obtains \( E_1 = R'_1 + R''_2 + R'_2 + R_1 \) after the error correction. It calculates a keying material \( KM \) with \( KM = D_2 + E_1 + R'_2 \), which is the same as the previous keying material \( KM = N + R'_2 + R'_1 \).

4. Our Proposals

In this section, we propose key establishment (KE1 and KE2) protocols using PUFs which satisfy two security requirements (i.e., Freshness and Non-Regeneration). The main idea of the KE1 and KE2 protocols is to make an attacker’s modifications on communication data not affect
calculating a keying material in order to prevent the key regeneration attack in Section 3.5.

Let Hash be a cryptographic hash function (e.g., SHA-2/3 [16], [17]) and let KDF be a secure key derivation function (e.g., [5]).

4.1. KE1 (Asymmetric Type)

The KE1 protocol proceeds as follows:

1) In order to let Module1 and Module2 establish the same fresh session key securely, Center sets a helper data \( D_1 = R_1 \) to Module2 where \( R_1 \) is a response of the PUF in the Module1 to a challenge \( C_1 \).

2) To establish a fresh session key with Module2, Module1 accepts the challenge \( C_1 \), obtains its response \( R_1' \) from its PUF, and then sends the syndrome (e.g., [28]) of \( R_1' \) (i.e., \( S_1' = R_1' \cdot H \)) to Module2. Also, Module1 calculates a keying material \( K_M \) with \( K_M = \text{Hash}(R_1') \).

3) After receiving \( S_1' \), Module2 applies the syndrome decoding algorithm to \( S_1' + D_1 \cdot H \) and then obtains \( E_1 = B_1 + R_1' \). It calculates a keying material \( K_M \) with \( K_M = \text{Hash}(D_1 + E_1) \).

4) Finally, both modules generate a session key \( SK = KDF(\text{Module1}, \text{Module2}, S_1', S_2', K_M) \).

4.2. KE2 (Symmetric Type)

The KE2 protocol proceeds as follows:

1) In order to let Module1 and Module2 establish the same fresh session key securely, Center sets a helper data \( D_2 = R_2 + N_2 \) and \( N_1 \) to Module1, and \( D_1 = R_1 + N_1 \) and \( N_2 \) to Module2, respectively. Here, \( R_1 \) and \( R_2 \) are responses of the PUFs in the modules to the respective challenges \( C_1 \) and \( C_2 \), and \( N_1 \) and \( N_2 \) are random code words of the underlying error correction code (which are binary vectors of the same length as \( R_1 \) and \( R_2 \)).

2) To establish a fresh session key with Module2, Module1 accepts the challenge \( C_1 \), obtains its response \( R_1' \) from its PUF, and then sends the syndrome of \( R_1' \) (i.e., \( S_1' = (N_1 + R_1') \cdot H \)), to Module2. In the same way, Module2 accepts the challenge \( C_2 \), obtains its response \( R_2' \) from its PUF, and then sends \( S_2' = (N_2 + R_2') \cdot H \) to Module1.

3) Module1 applies the syndrome decoding algorithm to \( S_2' + D_2 \cdot H \) and then obtains \( E_2 = R_2' + R_2 \). It calculates a keying material \( K_M \) with \( K_M = \text{Hash}(K_M1, K_M2) \) where \( K_M1 = R_1' + N_1 \) and \( K_M2 = D_2 + E_2 = R_2' + N_2 \). In the same way, Module2 applies the syndrome decoding algorithm to \( S_1' + D_1 \cdot H \) and then obtains \( E_1 = R_1' + R_1 \). It calculates a keying material \( K_M \) with \( K_M = \text{Hash}(K_M1, K_M2) \) where \( K_M1 = D_1 + E_1 = R_1' + N_1 \) and \( K_M2 = R_2' + N_2 \).

4) Finally, both modules generate a session key \( SK = KDF(\text{Module1}, \text{Module2}, S_1', S_2', K_M) \).

5. Discussions

5.1. Security

In this subsection, we show that the KE1 and KE2 protocols provide two security requirements (i.e., Freshness and Non-Regeneration).

Theorem 5.1. The KE1 and KE2 protocols satisfy Freshness under Assumption 1.

Proof. It is obvious from the descriptions of Section 4.1 and 4.2.

Theorem 5.2. Under Assumption 1 and 2, the KE1 and KE2 protocols satisfy Non-Regeneration if Hash is a cryptographic hash function.

Proof. Breaking Non-Regeneration of Module1 in the KE1 protocol and Module1 and Module2 in the KE2 protocol is as hard as finding a collision of the hash function Hash. According to Assumption 1 and 2, it is hard for an attacker to let the PUF generate exactly the same response. This guarantees that the inputs \( (R_1', \text{and } R_2') \) to the hash function Hash are distinct. If the attacker could regenerate the same \( K_M \), it would mean that a collision was found for the hash function Hash since the outputs of Hash are the same while the inputs are distinct.

5.2. Comparison

Here, we compare the KE1 and KE2 protocols (Section 4) with the previous PUF-based key exchange protocols [3], [4], [27].

In [27], Tuyls and Skoric proposed a PUF-based session key exchange protocol. Later, Busch et al., [4] showed an impersonation attack on Tuyls and Skoric’s protocol [27] when an attacker has access to the PUF for a short period of time, and then proposed PUF-based authentication and key establishment protocols using Bloom filters and hash trees. In [21], Rührmair et al., also showed a key regeneration attack on Tuyls and Skoric’s protocol [27] under the provision that an attacker gains physical access to the PUF twice. At CRYPTO 2011, Brzuska et al., [3] proposed PUF-based protocols for Oblivious Transfer (OT), Bit Commitment (BC) and Key Exchange (KE). In [19], Rührmair and Dijk gave a quadratic attack on Brzuska et al.,’s OT and BC protocols [3] if optical PUFs or electrical PUFs with challenge length of 64 bits are used. Also, Brzuska et al.,’s KE protocol [3] turned out to be insecure in the PUF re-use model, and in the combined PUF re-use and bad PUF model [20].

The KE1 and KE2 protocols of Section 4 are quite different from [3], [4], [27] in that:

- Freshness and Non-Regeneration of the KE1 and KE2 protocols are guaranteed under Assumption 1 and 2.
- The KE1 and KE2 protocols allow modules to establish a fresh session key without using a set (or
an exponential number) of PUF’s challenge-response pairs as in [3], [4], [27].

- The helper data \(D_1\) and/or \(D_2\) do not leave Module1 and Module2 all the time.
- The helper data in the KE2 protocol are composed of PUF responses and random code words of the underlying error correction code.

6. Conclusions

In this paper, we have proposed key establishment (KE1 and KE2) protocols using PUFs which can mitigate side-channel attacks and satisfy two security requirements (i.e., Freshness and Non-Regeneration) under Assumption 1 and 2. Also, we have discussed security of the KE1 and KE2 protocols, and compared with the previous PUF-based key exchange protocols [3], [4], [27].

Future works include formal security proofs for the KE1 and KE2 protocols, and their implementations along with experimental evaluations in terms of side-channel attacks.

Acknowledgment

We appreciate the anonymous reviewers’ constructive comments on this paper. This research was partly supported by JST, Infrastructure Development for Promoting International S&T Cooperation.

References


