6-6 Authentication Protocol and its Evaluation for IoT Devices

Daisuke MORIYAMA

Recently, many people argue that Internet of Things becomes a new information source near future. This project targets on establishing a cryptographically secure authentication protocol such that resource limited devices can validate the peer device. We also discuss how to implement the proposed protocol in software/hardware level via a FPGA board and show its result.

1 Introduction

IoT has become a major topic in a variety of industries, especially in the field of ICT, as one of the mainstream trends leading to the post-cloud computing age. The Security Architecture Laboratory has set the design of an anti-cyberattack architecture as one of the objectives in the midterm research planning starting from FY2011, capable of protecting a wide range of systems comprehensively — from the cloud to resource-limited terminals. Along this line, the author has conducted research and development, including joint study in cooperation with researchers overseas on such subjects as authentication protocols, typically the RFID tag that plays an important role in IoT terminals, and other application protocols.

The devices to be included in an IoT terminal can be typically classified into two categories: a sensor and an RFID tag. The major task assigned to the sensor is to gather electronic data — such as temperature, humidity, vibration — and then transmit the data to the server (through, for example, a sensor-to-sensor communication path). Research now underway on sensor security primarily focuses on data protection and retention of secrecy of location information. The main objective of an RFID tag, which is affixed to a "Thing," is to provide correct identification of it. As such, major security considerations include impersonation resilience and privacy protection.

In this study, the author developed an authentication protocol especially suited for IoT terminals that require a high level of security and privacy protection, and weighed proper steps to be taken for its implementation.

Although several research projects have been done in this area, none of them have provided their authentication protocols with rigorous security proof in terms of safety and privacy, nor implemented such protocols in a fashion that guaranteed sufficient safety and privacy [1]. The author conducted a study, in cooperation with Mr. Moti Yung (Google/Columbia University) and Associate Prof. Patrick Schaumont (Virginia Technical University), to combine two aspects of the challenge: theory and implementation. Specifically, we developed an authentication protocol characterized by provable security, and discussed the procedures to be followed for its implementation, including evaluation and analysis of constituent elements of the protocol. To realize provable security, the protocol takes advantage of a physically unclonable function (PUF), which utilizes the production tolerance of electronic circuits as a fingerprint to obtain a specific value. Based on this study, we finally evaluated our approach by implementing the software and hardware on an FPGA.

2 Construction of PUF-based authentication protocol

2.1 Constituent elements

In this paper, we make use of the following cryptologic functions.

- Random number generator: TRNG generates true random number sequences
- Physically unclonable function (PUF): The function $f: K \times D \rightarrow R$ determines the output $z \in R$ from a physical characteristic $x \in K$ and a message $y \in D$. The physical characteristic is basically determined based on the production tolerance of the IC circuit, so that each terminal constitutes a function that generates outputs dissimilar to others [2].
- Symmetric key encryption: $\mathsf{SKE} := (\mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ represents a symmetry key encryption system,

where SKE.Enc determines the ciphertext c from a secret key sk and plaintext m, and SKE.Dec restores the plaintext m from the secret key sk and ciphertext c.

- Pseudorandom function: PRF,PRF’: $K' \times D' \rightarrow R'$ outputs a random number and an indistinguishable bit series form the secret key $sk \in K'$ and message $m \in D'$.
- Fuzzy extractor: FE := (FE.Gen, FE.Rec) represents a fuzzy extractor. FE.Gen outputs a random number r and helper data hd from a variable input z. FE.Rec restores r, when a combination of the following two parameters is entered: z', which should lie in a short distance from z, and hd. The fuzzy extractor guarantees statistical indistinguishability between r and true random numbers even if hd is known, as long as the following conditions hold: the distance is no greater than d, and the minimum entropy of z is no smaller than $h\ (d, h)$. In many cases, fuzzy extractors are configured by combining error correction code and a randomness extractor[3].

2.2 Authentication safety model

This study assumes an IoT environment in which a server communicates with a plurality of devices (total number: num). The environment is supposed to have a privacy level that defies intervention from malicious adversaries and should not be susceptible to man-in-the-middle attacks. Particular concern should be exercised against the possibilities of eavesdropping and tampering of communication content on the assumption that even the authentication results and non-volatile memory storage can be threatened through physical attacks. An authentication protocol is said to be safe if, under such circumstances, it does not allow the server/terminal to accept any falsified or tampered authentication attempts that may have been generated by probabilistic polynomial time adversaries and man-in-the-middle attacks (data tampering in the middle of the communication route). In addition, the authentication protocol is said to satisfy privacy protection if it does not allow identification of the terminal from which an information leak takes place, even if all attempts are made to analyze the leaked information from terminals and communication lines.

2.3 Safe and privacy protective authentication protocol

Figure 1 shows the flow of the authentication process proposed by the author. The protocol is configured using a PUF. Because the PUF is specific to each terminal, the server must send an input y_1 to the terminal and receive a response z_1 from it in advance (for safety, this preliminary communication must take place offline).

In addition, the server sends a key, sk, to the server and stores the two parameters (sk and y_1) in non-volatile memory. In the next step of the authentication protocol, the terminal uses the PUF and fuzzy extractor to generate a random number r_1, then encrypts the helper data hd using sk while performing two-way authentication challenge and response using a pseudo random function PRF. The PRF also outputs the following keys: the key that acts as the random number used for XOR encryption of the PUF outputs that correspond to different inputs as well as for the message authenticator of entire messages, and the key that should be updated for the maintenance of security. Specifically, this approach has the following characteristics.

- Key extraction through the use of PUF:
  In the setup stage, the server stores a PUF output z_1. In the authentication phase, the terminal uses a physical characteristic value x_1 to seek $z'_1 \leftarrow f(x_i, y_1)$. Because it is not identical to z_1, the terminal seeks the helper data using a fuzzy extractor as $(r_1, hd) \leftarrow \mathrm{FE.Gen}(z'_1)$. The server decodes the helper data (encrypted before being sent) and determines it using the equation $r_1 \coloneqq \mathrm{FE.Rec}(z_1, hd)$, enabling both sides to extract the same (random) key. Thus, the use of PUF relieves the terminal of the need to store r_1 in its volatile memory. Even if a malicious

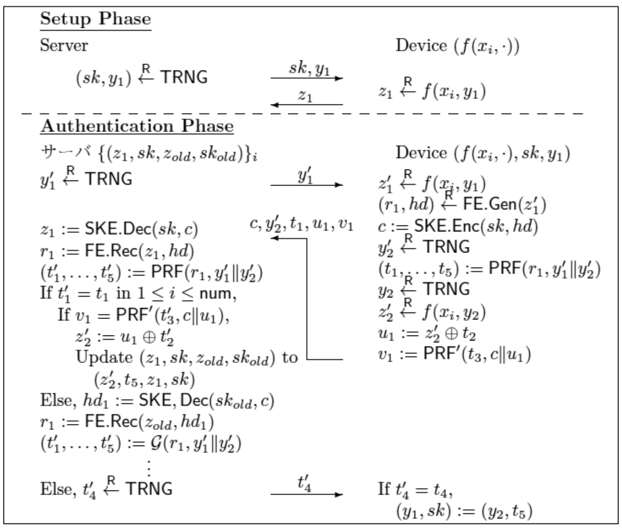

Fig. 1 Flow of the authentication protocol

adversary has a chance to peek into the volatile memory, the proposed protocol can nonetheless maintain safety.

- Two-way authentication and safe message transmission:
  After the key r_1 is successfully extracted on both sides, the pseudorandom function is run to determine the bit series $(t_1, \cdots, t_5)$. The elements of the bit series are used for the following purposes: (t_1, t_4) for two-way authentication, t_2 for XOR encryption of the PUF output, t_3 as the key for generating the value v_1 used to verify the entire message, and t_5 as the key for updating.
- Exhaustive search
  In protocol communication, terminals do not output specific values and IDs with a view toward protecting privacy. Instead, the server performs exhaustive search among the indices, $i \in \{1, \cdots, num\}$, in the database. Exhaustive search, albeit being inefficient, is essential to give extra consideration to privacy. and is a widely known approach in RFID authentication in general.

3 Implementation of constituent elements

For the protocol described in Section **2**, theoretical provability of its security can be demonstrated (see [4] for the details). However, separate considerations are needed as to the way in which it should be implemented. In this Section, the author discusses the length and handling method of each variable needed for implementation, whereby a 128-bit security level is assumed for evaluation.

3.1 Architecture

An FPGA board, SASEBO-GII (a Japanese product), is used as the implementation environment, which is a common practice in encryption studies. It is mounted with 2 Mbit SRAM (ISSI's IS61 LP6432 A) and 16 Mbit flash memory (ATMEL AT45 DB161 D). The SRAM is 64K memory with 32-bit output, and the flash memory is capable of communicating with the FPGA through an SPI connection.

3.2 SRAM PUF design

Much research has been done on PUF, and the SRAM PUF has been chosen for this study because of its highest cost efficiency. SRAM-type PUF is expected to provide the required characteristics of PUF: observing the state of SRAM at power up (indeterminate state before the execution of an active write operation) allows derivation of chip-specific values. To evaluate a PUF, estimate of entropy and the knowledge of noise for each output are required.

3.2.1 Entropy

In view of the fact that the output from PFU is not a true random number, entropy estimation is performed to obtain the measure of randomness. The data from 90 SASEBO-GIIs (each 2Mbit) was observed 11 times, and all the data (990 × 2 Mbit) were used for analysis.

The Shannon entropy ratio is calculated using the equation $\sum_{i=0}^{n} -b_i \log(b_i)/n \times 100$, where the data is divided into a series of n-bit blocks and each value is output with probability b_i. The calculation, with consideration given to the terminal-dependent variations, gave a range of ratio from 34-46%, which was independent of n. Shannon entropy represents the average amount of information. If the worst case scenario (i.e. minimum entropy ratio) is needed, the equation $n \times \min_i -b_i \log(b_i) \times 100$ should be used instead. The minimum entropy calculation on a bit-by-bit basis gave similar values to the Shannon entropy ratio, but the calculation on a byte ($n = 8$) basis gave a range from 5-15%. This deviation was caused by the fact that the value 0xAA was observed in many SRAMs. To circumvent this problem, each 32-bit data is divided into two 16-bit blocks and XORed with each other to balance out the bias. This modification resulted in higher minimum entropy ranges: the lower range limits for each terminal were no smaller than 26%.

3.2.2 Noise

Noise is an another factor that affects the use of PUF. For example, two identical observations of SRAM PUF do not necessarily yield the same data: superimposed noise causes small discrepancies. Use of error correcting code within the fuzzy extractor may be effective to avoid this problem, but it requires previous estimate of the noise occurrence frequency to determine appropriate parameters.

The XOR operation in the previous entropy processing inevitably increased the noise, which was evaluated to be, on average, 6.6 bits per each 64 bits. From this result, the amount of noise was assumed to be 10%. Measurement of the Hamming distance between any pairs of PUF resulted in 31.9 bits per 64 bits on an average, eliminating the possibility of confusing any two PUFs.

3.2.3 Usage as a random number generator

It is a common practice for a cryptographic protocol to use (cryptologically safe) random numbers, but, in the case of small-scale IoT devices, the generating element of the random number may require some cost. Our approach can make use of the noise associated with SRAM PUF (noise occurrence frequency 10% as described above) to generate a random number through repetitive XOR operations. In fact, the random number generated through XOR operations of 8 sets of data was verified to meet the requirements of the NIST randomness test [5]. In this case, a 128-bit random number can be generated from 1024-bit raw SRAM data. Because our protocol requires a 652-bit random number, the volume of raw SRAM data needed to generate it amounts to 5,216 bits.

3.3 Symmetry key encryption and pseudorandom function

Because the proposed authentication protocol is designed for use in IoT devices, we chose SIMON[6], a lightweight block cipher, as the symmetry key encryption. SIMON has gained higher reputation over other lightweight ciphers, and supports multiple safety levels. SIMON, as seen as a pseudorandom function, uses its encryption function Enc in CBC mode, in which the input message $(x_0, \cdots, x_n)$ is first converted to a plaintext block consisting of block ciphers, followed by entering the input size $|x|$ and counter. The counter is incremented until the required output length is obtained. The configuration of the implemented pseudo random number generator is shown in Fig.2.

3.4 Fuzzy extractor

3.4.1 Error correction code

Several methods have been investigated for the post-processing of the PUF data. In this study, we use a mechanism called "codeoffset," which uses BCH code. Assuming (BCH.Gen, BCH.Dec) a BCH code algorithm, data is restored in the following fashion:

- $\mathrm{Encode}(a)$: $\delta \leftarrow \mathrm{TRNG} \in \{0,1\}^{k_1}, cw := \mathrm{BCH.Gen}(\delta) \in \{0,1\}^{n_1}, hd := a \oplus cw$
- $\mathrm{Decode}(a', hd)$: $cw' := a' \oplus hd, cw := \mathrm{BCH.Dec}(cw'), a := cw \oplus hd$

The input a has been XOR-encrypted using a random number seed δ, eliminating the possibility of direct determination of a, even if hd information is leaked.

When (n_1, k_1, d_1)-BCH code is applied to PUF data, as the PUF output z_1 is divided into plural of n_1-bit blocks, exhaustive trials require the following number of computations.

$$2^{k_1 \cdot |z_1| / n_1}$$

This value must be larger than 128 bits. As the above analysis of SRAM PUF shows, the minimum entropy ratio is 26%. This indicates that if 504-bit data is divided into 8 blocks and (63, 16, 23)-BCH code is applied, it will contains $504 \times 0.26 > 128$-bit equivalent of random data.

Although the (63, 16, 23)-BCH code is capable of correcting $(23-11) / 63 \times 100 = 17.5\%$ of errors, the probability that 63-bit data may contain more than 12 error bits reaches 2.36% because each bit in SRAM PUF has 10% of noise. Thus, the probability that all 8 blocks are restored correctly will be no greater than $(1-0.0236)^8 \times 100 = 82.6\%$. To improve the probability, we adopted the following steps: original data is arranged in a matrix form and then commutated, followed by code-offset error correction using the same parameter. This approach improved the probability that all eight blocks are properly error corrected up to $1-1.92 \times 10^{-6}$, although two times as large as the helper data generated.

3.4.2 Randomness extractor

The randomness extractor is an algorithm used to extract a random number from a non-uniform array of bits (in this study, the array is generated by PUF). In this study, we adopted the pseudorandom function described above — i.e. symmetric key encryption based pseudorandom function — as the randomness extractor. Because the randomness extractor is inherently a probabilistic algo-

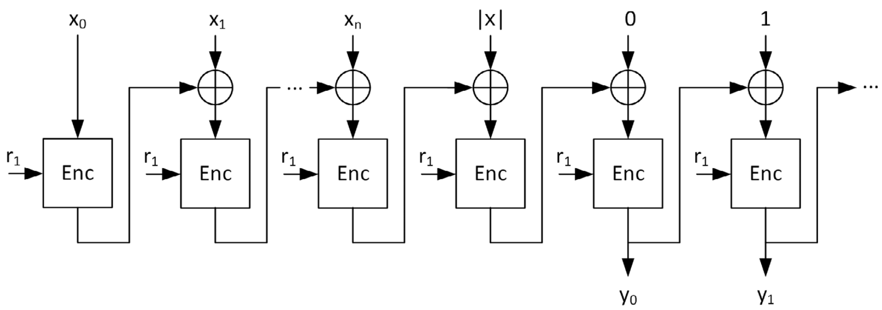

Fig. 2 Pseudorandom function that uses symmetry key encryption

Table 1 Data length and key length of the proposed protocol

Category	Purpose	Variable	64-bit security	128-bit security
Setup	Input address	y_1	12	12
	PUF's output	z_1	252	504
	Stored key	sk, sk'	64	128
Authentication Phase	PUF's output	z_1', z_2'	252	504
	Nonce	y_1', y_2'	64	128
	Randomness (fuzzy extractor)	δ, rnd	128	256
	PRF's secret	r_1	64	128
	Helper data	hd (includes rnd)	632	1,264
	Ciphertext	c	640	1,280
	PUF's input	y_2	12	12
	Mutual authentication	t_1, t_4	64	128
	XORed Element	t_2	252	504
	key for PRF′/MAC	t_3, s_1	64	128
	Update key	t_5	128	256
Communication	First message	y_1'	64	128
	Second message	(c, y_2', t_1, u_1, s_1)	1,084	2,168
	Third message	t_4'	64	128
Memory	Non-VM memory	(sk, sk', y_1)	140	268
	SRAM area for PUF		504	1,008
	SRAM area for RNG		2.656	5.216

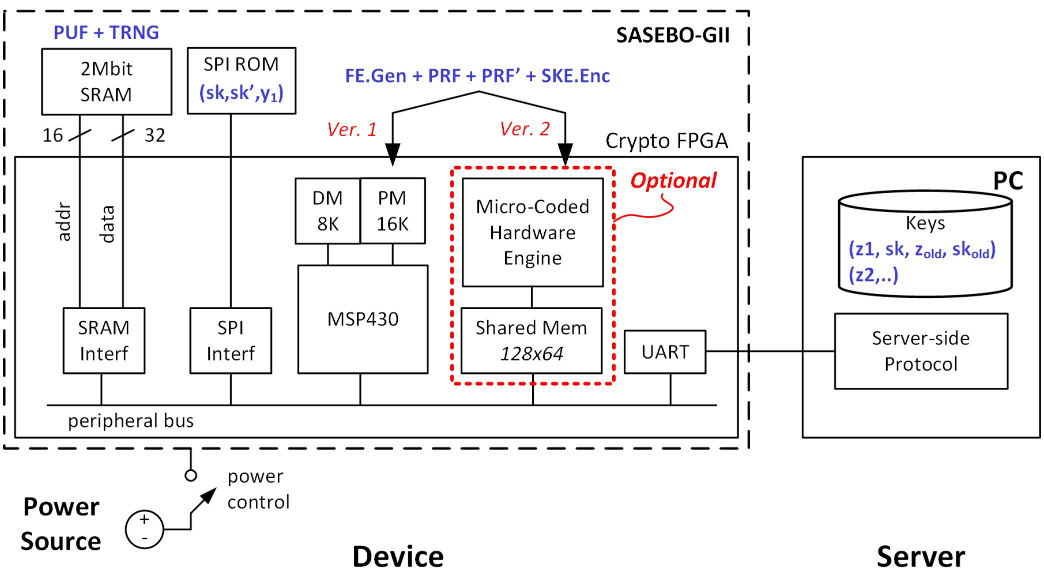

Fig. 3 The server and terminal architecture used in implementation evaluation

rithm, it requires a random number element. From the results of past research, at least twice as long a random number is required to maintain sufficient security as that dictated by the security level. Thus, a 256-bit random number is needed to guarantee 128-bit security.

To summarize the analysis described in this Section, required lengths of data and variables used in this protocol are listed in Table 1.

4 Architecture design

Figure 3 shows the system architecture — server and terminal — used for evaluating the proposed protocol. The system is implemented using a PC and SASEBO GII as emulators, and the terminal is realized by installing a micro-controller (MSP430) as a soft core inside the FPGA on SASEBO GII. On an as-needed basis, the encryption process is directly written to the FPGA (hardware engine) for comparison between hard- and soft-core implementation.

The system makes accesses to SRAM and non-volatile memory (EEPROM) mounted on the SASEBO GII as appropriate as the protocol proceeds. In the case of soft-core implementation, both the program memory and data memory reside in MSP430: the encryption protocol is written in the program memory, and variable values are stored in the data memory. The server side has a database that contains such information as the secret keys and PUF outputs obtained from the terminal. In the implementation of this study, the server and terminal communicate with each other through a USB serial link.

The hardware engine contains such procedures as the encryption steps according to SIMON, calculations performed by pseudorandom function that makes use of SIMON, and BCG code calculations. The hardware engine and MSP430 exchange data through the medium of common memory. When the hardware engine is used, necessary information is copied from MSP430 memory to the common memory before activating the above-described processes on the hardware. Then, the results are written in the common memory, allowing access from MSP430. This method entails overhead determined by the volume of communication, but, as shown in the next Section, the communication works faster than the all-software implementation.

Table 2 Memory footprint (byte) in MSP430

Category	64-bit MSP430	128-bit MSP430	128-bit MSP430 + HW
HW extraction	1,022	1,022	1,398
Communication	496	644	628
SIMON	1,604	2,440	0
BCH code	1,214	1,214	0
PUF + FE	562	646	590
RNG	396	456	396
Protocol	1,568	1,682	1,908
Text	6,862	8,104	4,920
Variable	424	656	656
Constant	197	197	73
Data	621	853	729

5 Implementation evaluation

In this Section, we evaluate the following items using the actual implementation of the system: the cost associated with the terminal implementation, and calculation complexity. Three cases were examined in this study: two software implementations (64-bit security and 128-bit security) on MSP430, and a hardware engine (128-bit security).

5.1 Implementation cost

Memory usage for the three cases is summarized in Table 2 (object code and data memory usage on MSP430 included). GNU gcc compiler (ver.4.6.3, optimization level 2) was used to produce object code for MSP430. MSP430 is mounted with 8 KB memory, enough volume to allow all three implementations.

5.2 Computational complexity

Table 3 shows the comparison of computational complexity (in system clock unit) for three different implementations of the protocol. In view of the resource-limited IoT devices, MSP430 was run at 1.846 Mhz. Computational complexity was drastically reduced in the hardware engine-based implementation. Note here that the figures in the Table include the time required to transfer data to the hardware engine (actual computation for the encryption process took 4,486 clocks).

Table 3 Calculation complexity of the proposed protocol (cycles)

Protocol Steup	Implementation Target	64-bit	128-bit	128-bit with HW
Read sk, sk', y_1	Read ROM	31,356	61,646	61,646
$y_2' \xleftarrow{R} \mathsf{TRNG}$, $y_2 \xleftarrow{R} \mathsf{TRNG}$	SRAM TRNG	11,552	23,341	22,981
$z_1' \xleftarrow{R} f(x_i, y_1)$,$z_2' \xleftarrow{R} f(x_i, y_2)$	SRAM PUF	4,384	9,082	8,741
$(r_1, hd) \xleftarrow{R} \mathsf{FE.Gen}(z_1')$	BCH Encoder Strong extractor	268,820 28,691	485,094 205,080	18,597
$(t_1, \ldots, t_5) := \mathsf{PRF}(r_1, y_1' \| y_2')$	PRF	44,355	299,724	
$c := \mathsf{SKE.Enc}(sk, hd)$	Encryption	39,583	252,829	
$v_1 := \mathsf{PRF}'(t_3, c \| u_1)$	PRF′	57,601	394,126	
Overall		486,343	1,730,922	111,965
Write y_2, t_5	Write ROM	76,290	128,829	128,849

6 Concluding remarks

In this report, the author explained the process of putting an anonymous authentication protocol for IoT devices into practice — from the theoretical foundation to software/hardware implementation — as well as its evaluation. Evaluation of the authentication scheme as a whole, with all the constituent elements of the protocol implemented in it, provides an important viewpoint for verifying future operability in actual terminals. This study places focus on the implementation method of the protocol, and indicates that there is yet room for improvement. Viewed from the architecture level, for example, there seems to be room for further item-by-item optimization -e.g. computational complexity, implementation cost, and power consumption. Note that another implementation approach — using a different PUF, lightweight symmetric key encryption, and error correction code — can lead to different results. Comparative evaluation of these approaches enables deriving the optimum authentication protocol, which will be implemented in a variety of IoT devices provided by private sector companies in the future. The author hopes the protocol will contribute to the realization of an ICT society with due considerations given to user security and privacy.

References

1 Delvaux,J., Gu,D., Peeters,R., and Verbauwhede,I., "A survey on lightweight entity authentication with strong PUFs," IACR Cryptology ePrint Archive 2014/977, 2014.

2 Maes,R., "Physically Unclonable Functions - Constructions, Properties and Applications," Springer, 2013.

3 Dodis,Y., Ostrovsky,R., Reyzin,L., and Smith,A., "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," SIAM J. Comput. 38(1), pp.97–139 , 2008.

4 Aysu,A., Gulcan,E., Moriyama,D., Schaumont,P., and Yung,M., "End-to-end design of a PUF-based privacy preserving authentication protocol," In: CHES 2015. LNCS, vol.9293, pp.556–576. Springer, Heidelberg. Full version is available at IACR Cryptology ePrint Archive 2015/937, 2015.

5 Rukhin,A., Soto,J., Nechvatal,J., Smid,M., Barker,E., Leigh,S., Levenson,M., Vangel,M., Banks,D., Heckert,A., Dray,J., and Vo,S., "A statistical test suite for the validation of random number generators and pseudo random number generators for cryptographic applications," Special Publication 800-22 Revision 1A, April, 2010.

6 Beaulieu,R., Shors,D., Smith,J., Treatman-Clark,S., Weeks,B., and Wingers, L., "The SIMON and SPECK families of lightweight block ciphers," IACR Cryptology ePrint Archive 2013/404, 2013.

Daisuke MORIYAMA, Ph.D.

Former Researcher. Security Architecture Laboratory, Network Security Institute
Cryptographic Protocol