Advances in Engineering Research, volume 113
International Forum on Mechanical, Control and Automation (IFMCA 2016)

IPLDS: Integrity Protection for Long Data Streams in Wireless Sensor Networks Using Montgomery Modular Multiplication

Daoli Huang[1, a] and Changsheng Wan[2,b]

[1]Key Lab of Information Network Security of Ministry of Public Security of China, Shanghai, 201204, China

[2]Radio Department, Southeast University, Nanjing, Jiangsu, 210096, China

[a]huangdaoli@stars.org.cn, [b]wan.changsheng@163.com

Keywords: Integrity protection, Long data stream, Montgomery modular multiplication.

Abstract. Due to limited energy of sensors, existing data integrity protection mechanisms for wireless sensor networks (WSNs) typically use a simple hash function for signing and verification. However, since the computation cost of hash function depends on the length of the input data, this cost may be high for long data streams. In this paper, we present a secure and efficient data integrity protection protocol for WSNs named IPLDS, which only uses hash function for processing a short block of data, and mainly operates on the long data stream using Montgomery modular multiplication, which can reduce the computation costs of long data streams significantly.

Introduction

Nowadays, WSNs have been widely deployed in military, environmental and other commercial applications [1]. To transport data between two sensor nodes over a hostile network, integrity protection modules have been deployed, ensuring that the transmitted data isn't tampered by the attackers over the network. Regardless of the technology implemented, as shown in Fig.1, the data integrity protection scenario includes three parties: the base station (BS), the sender and the receiver [2] [3]. Before data-transmitting, the sender and the receiver (i.e. two wireless sensor nodes) are deployed with a shared secret by the BS. During data-sending process, the sender organizes data readings into elements of fixed size [4], and generates a message authentication codes (MAC) [5] for each data element, using the shared secret. Then it sends data elements and their MACs to the receiver. Upon receiving data elements and MACs, the receiver verifies MACs to ensure that the received data isn't tampered, using the shared secret.

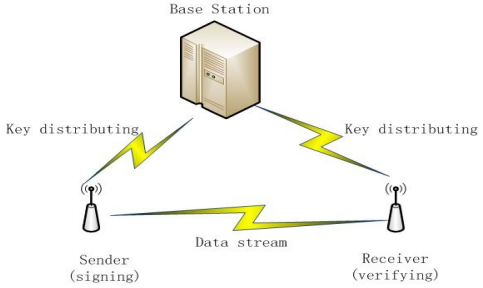

Fig.1. Data integrity protection overview.

Computation cost is a serious concern for the above data integrity protection system. Due to limited energy, the sender and the receiver are deeply concerned about the high computation costs arising from signing and verifying long data streams. Therefore, to increase the lifetime of sensor nodes, current data integrity protection techniques mainly employ a simple hash function for generating and verifying MACs, instead of public key cryptography. Unfortunately, in current schemes, the length of hash function's input depends on that of the data stream, resulting in high computation costs and short lifetime for long data streams. Therefore, it is prerequisite to elaborately

design a data integrity protection protocol for sensor networks, in which the length of the hash function's input is independent of the length of data streams.

Obviously, designing a data integrity protection protocol for WSNs is a nontrivial task, because the resource-restrained sensor nodes are not competent to sign, transmit and verify long data streams. When considering this research issue, we observe that none of the existing cryptographic primitives can be directly applied to achieve the goals discussed above. This becomes a more severe issue given the trend that more and more sensor networks are being deployed.

Motivated by this observation, we propose a novel approach to ensure data integrity in WSNs called IPLDS, which is built on hash function. However, different from current hash function based mechanisms, IPLDS provides a novel signing algorithm. We don't use traditional signing algorithm here due to the following reasons: In current signing algorithm, the hash function operates on the whole data stream, which is quite time-consuming. On the other hand, observing that MontMM [6] is much more efficient than hash function, IPLDS aims to mainly perform MontMM operations on the data stream, and use hash function only for processing a short block of data. By doing so, the computation costs on both the sender and the receiver can be reduced significantly.

IPLDS: The Protocol

The Single-data-element Scenario

The basic protocol includes three phases, as shown below.

(1) The pre-configuration phase.

During this phase, the BS generates a set of shared secret keys and distributes it to the sender and the receiver, respectively. And the key distributing channels between the BS and the sender/receiver should provide integrity/confidentiality/replay-resistance protections, so that the sender and the receiver can ensure that they get the correct keys and the keys are not leaked to attackers. The key generating algorithm is shown below.

$sk \leftarrow Genkey(1^l)$. This algorithm is run by the BS. It takes as input the security parameter l, and output a random prime p and two random shared secret keys $\{b_0, b_1\}$ for the sender and the receiver. In this algorithm, l is the length of $\{b_0, b_1\}$ which determines the security level of IPLDS (Typically, l should be at least 80 to achieve a primary security level), and p determines the finite field of data elements and keys (e.g. $b_0, b_1 \in Z_p$) which should be more than l -bit.

After the pre-configuration phase, the sender and the receiver hold $sk = \{b_0, b_1, p\}$ for signing and verifying, respectively.

(2) The signing phase.

During this phase, the sender generates a MAC for the data element $d \in Z_p$, using the following signing algorithm.

$t \leftarrow Gensig(d, sk)$. Given $d \in Z_p$ and $sk = \{b_0, b_1 \in Z_p\}$, the sender splits d into d_f and d_l, and computes the MAC for d as $t = SHA1(b_0 \| sID \| rID \| d_f) + b_1 d_l \bmod p$, where sID is the identity of the sender and rID is the identity of the receiver. Note that the length of $b_0 \| sID \| rID \| d_f$ should be less than 448-bit, so that SHA1's input always forms one block.

Then the sender sends (sID, rID, d, t) to the receiver over the hostile network.

(3) The verifying phase.

Upon receiving (sID, rID, d, t) from the sender, the receiver (rID) first checks (sID, rID) to make sure this data is sent from the sender (sID), and should be processed by itself. Then it verifies (d, t) using the $Gensig$ algorithm to ensure d isn't tampered by an attacker.

From the above *Gensig* algorithm, it can be seen that the length of the hash function's input is less than 448-bit (i.e. 1 block), and is independent of the length of d. Moreover, for a long data element (i.e. $d_l > 0$), this signing algorithm processes d_l using MontMM. On the other hand, in pure hash function based schemes, signing algorithm processes d_l using a hash function. Therefore, since MontMM is much more efficient than hash functions, IPLDS can reduce the signing and verifying costs significantly.

The Multiple-data-element Scenario

The multiple-data-element scenario is slightly different from the single-data-element scenario, as shown below:

In the pre-configuration phase, the BS generates a set of keys for the sender and the receiver as $sk = \{b_0, b_1, ..., b_s \in Z_p, p\}$.

In the signing phase, the sender splits the data stream into s elements ($d_1, ..., d_s \in Z_p$), and generates one signature for multiple data elements as $t = SHA1(b_0 \| sID \| rID) + \sum_{i=1}^{s} b_i d_i \bmod p$.

In the verifying phase, the receiver verifies (d, t) as $t = SHA1(b_0 \| sID \| rID) + \sum_{i=1}^{s} b_i d_i \bmod p$.

From the above signing/verifying algorithm for the multiple-data-element scenario, it can be seen that: (i) s data elements share one $SHA1$ hash function, while each data element consumes only one MontMM. (ii) the length of hash function's input is less than 448-bit (i.e. 1 block). On the other hand, in pure hash function based schemes, each data element consumes one hash function. Since MontMM is much more efficient than $SHA1$, IPLDS can reduce the signing and verifying costs significantly.

Security Analysis

Proof of the security of the single-data-element scenario.

Part 1) The security assumption.

The hash function assumption (HF). We assume that the hash function used in our scheme (i.e. $SHA1$) is secure. That is, given sID, rID and d_f for randomly-distributed unknown b_0, there is no t -time algorithm, which has the non-negligible probability e in computing $SHA1(b_0 \| sID \| rID \| d_f)$.

Part 2) The adversary model.

To satisfy requirement (1), it must be ensured that no one on the hostile network can tamper the data. Therefore, the potential adversary is a malicious node on the hostile network. It holds $\{sID, rID, d\}$. Given $\{sID, rID, d, t, d' \neq d\}$, the adversary outputs t'. We say that the adversary wins the game if $t' = SHA1(b_0 \| sID \| rID \| d_f') + b_1 d_l' \bmod p$.

Part 3) The security reduction.

This part shows that the security of IPLDS depends on the security assumption (i.e. the HF assumption defined in part 1), as shown in Theorem 1.

Theorem 1. If there exists an adversary that can forge a signature for the tampered $d' \neq d$ with the probability e, the simulator can solve the HF with the probability e.

Proof. Given sID, rID and d_f for randomly distributed unknown b_0, the simulator can compute $SHA1(b_0 \| sID \| rID \| d_f)$ in the following three steps:

Step 1) The simulator runs the adversary with the parameter $\{sID, rID, d_f, d_l, t, d_f' = d_f, d_l' \neq d_l\}$. Let the output be t_j'. The probability that $t' = SHA1(b_0 \| sID \| rID \| d_f) + b_1 d_l' \bmod p$ is e.

Step 2) Since $t = SHA1(b_0 \| sID \| rID \| d_f) + b_1 d_l \bmod p$, the simulator computes $b_1 = (t' - t)(d_l' - d_l)^{-1} \bmod p$.

Step 3) After getting b_1, the simulator computes $SHA1(b_0 \| sID \| rID \| d_f) = t - b_1 d_l = t - (t' - t)(d_l' - d_l)^{-1} d_l \bmod p$.

From the above proof, it can be seen that the probability (i.e. the advantage) for computing $SHA1(b_0 \| sID \| rID \| d_f)$ is the multiplication of the probabilities of the three steps: $e \times 1 \times 1 = e$.

Efficiency Evaluation

All our experiments were conducted on an Intel i7 processor system at 3.40 GHz, using the CentOS operating system. We investigated the time costs of SHA1 and MontMM using the OPENSSL library [7] in Table 1 and 2, respectively. To achieve the 80-bit security parameter, we generate an 80-bit random number as the signing key, which acts as one of the multipliers of MontMM. Therefore, in table 2, one multiplier of MontMM is 80-bit, while the other has the same length as the modular p.

TABLE 1 Time Costs of SHA1
(Interval: 64-byte=512-bit=one SHA1 block; Unit: $10^{-2} ms$)

	T_{56}	T_{120}	T_{184}	T_{248}	T_{312}	T_{376}
Time costs	56	99	139	181	223	267

T_{56}, T_{120}, T_{184}, T_{248}, T_{312} and T_{376} are time costs for 56-byte, 120-byte, 184-byte, 248-byte, 312-byte and 376-byte input data, respectively.

TABLE 2 Time Costs of MontMM
(Interval: 10-byte=80-bit; Unit: $10^{-2} ms$)

	T_{10}	T_{20}	T_{30}	T_{40}	T_{50}	T_{60}
Time costs	3.2	5.0	7.2	10.1	17.8	20.9

T_{10}, T_{20}, T_{30}, T_{40}, T_{50} and T_{60} are time costs for 10-byte, 20-byte, 30-byte, 40-byte, 50-byte and 60-byte p (i.e. the modular), respectively.

From table 1, it can be seen that: (i) the time cost of SHA1 for the first data block (i.e. the first448-bit data) is $56 \times 10^{-2} ms$. (ii) for each subsequent data block (i.e. 512-bit), the time cost of SHA1 is around $(40-50) \times 10^{-2} ms$ (e.g. $T_{120} - T_{56} = 99 - 56 = 43$). That is to say, the time cost for processing the first block is slightly higher. In addition, performing the linear fitting operation on table 1 using the software MATLAB, we get the following byte-level equation: $T_{SHA1} = 0.65Len + 19.6$, where Len is the data length in bytes, T_{SHA1} is the time cost of SHA1, and the unit is $10^{-2} ms$.

From table 2, it can be seen that: (i) for less than 40-byte data blocks, the time cost of MontMM for each 10-byte data is $(2-3) \times 10^{-2} ms$. (ii) when the data length is more than 40-byte, the time cost will increase rapidly (e.g. $T_{50} > 5T_{10}$ and $T_{60} > 6T_{10}$). Therefore, in the single-data-element scenario, when the sender splits the long data element d into multiple 80-bit $d_1, ..., d_s \in Z_p$ for generating short signatures, it will not increase the total computation cost. Observing this property, we can simply use T_{10} for computing the time costs of MontMM with different modular lengths, and get the following byte-level equation: $T_{MM} = Len * T_{10} / 10 = 0.32Len$, where Len is the data length in bytes, T_{MM} is the time cost of MontMM, and the unit is $10^{-2} ms$.

Combing the results of table 1 and 2, it can be seen that: (1) for short input data (e.g. 10-byte data blocks), the time cost of MontMM is less than 10% to that of SHA1. For example,

$T_{10} / T_{56} = 3.2 / 56 = 5.7\% < 10\%$. (2) for long input data, the time cost of MontMM is around 50% to that of SHA1, because $\lim_{Len\to\infty} T_{mm} / T_{SHA1} = \lim_{Len\to\infty} 0.32Len / (0.65Len + 19.6) = 0.5$ (i.e. 50%).

In summary, the computation cost of MontMM is around 5.7%-50% to that of SHA1.

Conclusion

In this paper, we have identified the characteristics of data integrity protection in wireless sensor networks, and concluded 4 properties that a secure and efficient integrity protection scheme should satisfy. Moreover, we have proposed a novel protocol named IPLDS. The protocol satisfies a set of important requirements which have not been addressed by earlier works. The security analysis shows the proposed approach is feasible for real applications, and the experimental results show that it is much more efficient than current schemes.

Acknowledgment

This paper is supported by the NSFC (No.61101088,No.71402070), the NSF of jiangsu province (No.BK20161099), and the Opening Project of Key Lab of Information Network Security of Ministry of Public Security (No. C16604).

References

[1] Xiangqian Chen, Kia Makki, Kang Yen, and Niki Pissinou, “Sensor Network Security: A Survey”, IEEE COMMUNICATIONS SURVEYS & TUTORIALS, VOL. 11, NO. 2, SECOND QUARTER 2009.
[2] Perrig, A.; Przydatek, B.; Song, D. SIA: Secure Information Aggregation in Sensor Networks. J. Comput. Secur. 2007, 15, 69-102.
[3] Perrig, A.; Szewczyk, R.; Wen, V.; Culler, D.; Tygar, J.D. SPINS: Security Protocols for Sensor Networks. In Proceedings of the 7th Annual International Conference on Mobile Computing and Networking, Rome, Italy, July 2001.
[4] Albath, J.; Madria, S. Practical Algorithm for Data Security (PADS) in Wireless Sensor Networks. In Proceedings of the 6th ACM International Workshop on Data Engineering for Wireless and Mobile Access, Beijing, China, June 2007; pp. 9-16.
[5] A. Menezes and et al, "Handbook of Applied Cryptography," CRC press, Dec. 1996.
[6] P.L. Montgomery, “Modular Multiplication without Trial Divi-sion,” Math. Of Computation, vol.44, no.170, pp.519-521, Apr.1985.
[7] Openssl.org, “openssl-1.0.1e.tar.gz” Feb 2013. [Online]. Available: http://www.openssl.org/source/