DTLS-HIMMO: Achieving DTLS Certificate Security with Symmetric Key Overhead

Oscar Garcia-Morchon(✉), Ronald Rietman, Sahil Sharma, Ludo Tolhuizen, and Jose Luis Torre-Arce

Philips Group Innovation, Research, Eindhoven, The Netherlands
{oscar.garcia,ronald.rietman,sahil.sharma,ludo.tolhuizen, jose.luis.torre.arce}@philips.com

Abstract. Billions of devices are being connected to the Internet creating the Internet of Things (IoT). The IoT not only requires strong security, like current Internet applications, but also efficient operation. The recently introduced HIMMO scheme enables lightweight and collusion-resistant identity-based key sharing in a non-interactive way, so that any pair of Internet-connected devices can securely communicate.

This paper firstly reviews the HIMMO scheme and introduces two extensions that e.g. enable implicit credential verification without the need of traditional digital certificates. Then, we show how HIMMO can be efficiently implemented even in resource-constrained devices, enabling combined key agreement and credential verification more efficiently than using ECDH-ECDSA. We further explain how HIMMO helps to secure the Internet and IoT by introducing the DTLS-HIMMO operation mode. DTLS, the datagram version of TLS, is becoming the standard security protocol in the IoT, although it is very frequently discussed that it does not offer the right performance for IoT scenarios. Our design, implementation, and evaluation show that DTLS-HIMMO operation mode achieves the security properties of the DTLS-Certificate security suite while exhibiting the overhead of symmetric-key primitives without requiring changes in the DTLS standard.

Keywords: HIMMO · Lightweight · (D)TLS · Quantum · TTP infrastructure.

1 Introduction

The Internet of Things (IoT) is connecting billions of smart devices deployed in critical applications like healthcare, distributed control systems, smart cities and smart energy. The IoT not only needs strong security solutions, like today's Internet, but also efficient approaches to secure the data exchanges between smart devices, and between smart devices and the Internet.

The Transport Layer Security (TLS) [5] and its Datagram version (DTLS) are two of the most important protocols used to secure the Internet. DTLS is becoming the security standard to secure the IoT since it is required by many Machine to Machine standards such as LWM2M. However, it is very frequently

G. Pernul et al. (Eds.): ESORICS 2015, Part I, LNCS 9326, pp. 224–242, 2015.
DOI: 10.1007/978-3-319-24174-6_12

discussed that DTLS and its cipher suites are too heavy for many IoT use cases. In some cases, resource limitations (e.g., memory or energy) of end devices may prohibit the support of the standard algorithms. In other cases, the large number of devices and lack of user interface make the managing of large amounts of credentials for all those devices extremely complex. In some situations, devices are managed over a cellular connection and each extra byte of consumed bandwidth incurs costs. It is estimated that currently 70 % of the IoT devices have security risks and are often poorly managed [1]. At the same time, the advent of quantum computers will endanger all key agreement primitives used in (D)TLS except pre-shared keys [3]. Thus, there is a need for a solution that is secure, efficient, scalable, simple to use, and if possible, quantum-secure.

The HIMMO scheme [10] is a fully-collusion resistant key pre-distribution scheme that enables lightweight identity-based key sharing between devices in a single message, which is ideal for real-time IoT interactions. This paper builds on the HIMMO scheme and describes a couple of extensions, e.g., enabling implicit credential verification without the need of traditional digital certificates. Next, we show how HIMMO can be efficiently implemented even on resource-constrained devices. We further put HIMMO in the context of the IoT and describe the design, implementation, and evaluation of the (D)TLS-HIMMO operation mode as a lightweight alternative to existing public-key based solutions. This new operation mode for (D)TLS allows us to achieve security properties of a (D)TLS-certificate exchange – key agreement, mutual authentication of client and server, and verification of credentials – with the resource needs of symmetric-key primitives.

The rest of this paper is organized as follows. Section 2 describes the features of IoT scenarios, security needs, and relevant IoT security standards. Section 3 reviews the HIMMO scheme. Section 4 presents an efficient algorithm for key agreement and performance results. Section 5 introduces the (D)TLS-HIMMO operation mode. In Sect. 6, we compare DTLS-HIMMO with existing (D)TLS alternatives. Section 7 concludes this paper and discusses future work.

2 Preliminaries

We consider a network of low-resource devices that should be capable to pairwise communicate with each other. In order to secure this communication, each pair of devices should be able to generate a common key. In the HIMMO scheme and in the extensions we discuss in this paper, one or more Trusted Third Parties (TTPs) provide all devices with secret information, termed keying material, that will be used in generating such common keys. It is assumed that the TTP can provide the keying material in a secure manner.

2.1 Security Standards in the Internet (of Things)

The Internet is protected by two main standard protocols, the Internet Protocol Security (IPSec) and the Transport Layer Security (TLS). IPSec offers security at network layer while TLS protects exchange of information between applications at transport layer. Both IPSec and TLS have an initial phase enabling authentication of peers, agreement on a session key, negotiation on the ciphersuite, etc. Afterwards, the data flow can be secured in the sense of confidentiality, authenticity, integrity and freshness by making use of the agreed session keys.

The TLS protocol runs on top of TCP and is used to secure our HTTP Internet connections when we access the bank online, we do the tax computation, or when we access some healthcare services. The Internet is further evolving to connect many smart objects creating the Internet of Things (IoT) comprising smart meters, healthcare devices, etc. In a typical use case, devices communicate end-to-end with a back-end server, reporting information such as energy consumption, maintenance data, etc. by means of protocols such LWM2M that are protected by Datagram Transport Layer Security (DTLS), the equivalent of TLS running on UDP. Note that DTLS builds on TLS, and therefore both protocols are very similar, the only differences are a few extensions ensuring that protocol can run on UDP. There are more than 200 known cipher-suites for TLS, e.g. see [2]. OpenSSL is one of the most common and used libraries implementing TLS and most of its different cipher-suites. For the Internet of Things, other libraries such as CyaSSL[1] are also popular due to their smaller footprint and simple API supporting more than 70 cipher-suites.

2.2 DTLS-PSK

Pre Shared Key (PSK) mode is a set of ciphersuites applicable to both TLS and DTLS [6]. Although not in common use on the Internet, (D)TLS-PSK is widely employed by IoT devices since it has very low resource needs. The ciphersuite TLS_PSK_WITH_AES_128_CCM_8 [13], for instance, uses PSK as the authentication and key exchange algorithm.

In DTLS-PSK, both clients and servers may have pre-shared keys with different parties, the client indicates which key to use with the PSK-identity in the *ClientKeyExchange* message. The server may help the client in selecting the identity to use with the *PSK-identity-hint* in the *ServerKeyExchange* message. For IoT devices, the PSK identity can be based on the domain name of the server and, thus, the *PSK-identity-hint* need not be sent by the server [17], so the *ServerKeyExchange* is optional. The credentials (the pre-shared keys themselves) are stored as part of hardware modules, such as SIM cards, and sometimes, on the firmware of resource-constrained devices themselves. The session keys for the DTLS record session are derived from the PSK using the TLS Pseudo Random Function (PRF) as defined in [5]. The cookie exchange is used to prevent denial of service attacks on the server. The Constrained Application Protocol (CoAP) [16] mandates the use of TLS_PSK_WITH_AES_128_CCM_8 for the use with shared secrets [17].

[1] Cyassl. http://www.yassl.com/yaSSL/Products-cyassl.html.

2.3 Attack Model and Security Goals

With the HIMMO-DTLS extension, we aim at ensuring all security properties of DTLS-PSK [6] including also the capability of credential verification. This work does not provide other security features such as perfect forward secrecy or non-repudiation. We assume that the TTP can securely distribute the keying materials to the devices. However, an attacker can later monitor, eavesdrop, and modify message exchanges. We further assume that the attacker can compromise arbitrary devices and extract secret keying material which he can combine to attack the system.

3 HIMMO and HIMMO Extensions

HIMMO is a Key Pre-Distribution Scheme (KPS), a concept introduced by Matsumoto and Imai in 1987 [12]. An elegant and efficient KPS, based on symmetric polynomials, has been introduced by Blundo *et al.* [4]. There was no KPS that is both efficient and not prone to efficient attacks of multiple colluding (or compromised) nodes, see the references in [9], until recently the HIMMO scheme solved this problem. This section reviews the operation of the HIMMO scheme that enables any pair of devices in a system to directly agree on a common symmetric-key based on their identifiers and a secret key generating polynomial as introduced in [10]. The underlying security principles on which HIMMO relies have been analyzed in [7,8]. Furthermore, this section describes two protocol extensions of the HIMMO scheme as described in [9].

We use the following notation: for each integer x and positive integer M, we denote by $\langle x\rangle_M$ the unique integer $y \in \{0, 1, \ldots, M-1\}$ such that $x \equiv y \mod M$.

3.1 HIMMO Operation

Like any KPS, HIMMO requires a trusted third party (TTP), and three phases can be distinguished in its operation [12].

In the **setup phase**, the TTP selects positive integers B, b, m and α, where $m \geq 2$. The number B is the bit length of the identifiers that will be used in the system, while b denotes the bit length of the keys that will be generated. The TTP generates the public modulus N, an odd number of length exactly $(\alpha+1)B+b$ bits (so $2^{(\alpha+1)B+b-1} < N < 2^{(\alpha+1)B+b}$). It also randomly generates m distinct secret moduli $q_1, \ldots, q_m$ of the form $q_i = N - 2^b\beta_i$, where $0 \leq \beta_i < 2^B$ and at least one of $\beta_1, \ldots, \beta_m$ is odd. Finally, the TTP generates the secret root keying material, that consists of the coefficients of m bi-variate symmetric polynomials of degree at most α in each variable. For $1 \leq i \leq m$, the i-th root keying polynomial $R^{(i)}(x, y)$ is written as

$$R^{(i)}(x,y) = \sum_{j=0}^{\alpha}\sum_{k=0}^{\alpha} R^{(i)}_{j,k} x^j y^k \text{ with } 0 \leq R^{(i)}_{j,k} = R^{(i)}_{k,j} \leq q_i - 1. \tag{1}$$

In the **keying material extraction phase**, the TTP provides each node ξ in the system, with $0 \leq \xi < 2^B$, the coefficients of the key generating polynomial G_ξ:

$$G_\xi(y) = \sum_{k=0}^{\alpha} G_{\xi,k} y^k \tag{2}$$

where

$$G_{\xi,k} = \langle \sum_{i=1}^{m} \langle \sum_{j=0}^{\alpha} R_{j,k}^{(i)} \xi^j \rangle_{q_i} \rangle_N . \tag{3}$$

In the **key generation phase**, a node ξ wishing to communicate with node η with $0 \leq \eta < 2^B$, computes:

$$K_{\xi,\eta} = \langle \langle G_\xi(\eta) \rangle_N \rangle_{2^b} \tag{4}$$

It can be shown that $K_{\xi,\eta}$ and $K_{\eta,\xi}$ need not be equal. However, as shown in Theorem 1 in [9], for all identifiers ξ and η with $0 \leq \xi, \eta \leq 2^B$,

$$K_{\xi,\eta} \in \{\langle K_{\eta,\xi} + jN \rangle_{2^b} \mid 0 \leq |j| \leq 2m\}. \tag{5}$$

In order to perform key reconciliation, i.e. to make sure that ξ and η use the same key to protect their future communications, the initiator of the key generation (say node ξ) sends to the other node, simultaneously with an encrypted message, information on $K_{\xi,\eta}$ that enables node η to select $K_{\xi,\eta}$ from the candidate set $C = \{\langle K_{\eta,\xi} + jN \rangle_{2^b} \mid 0 \leq |j| \leq 2m\}$. No additional communication thus is required for key reconciliation. The key $K_{\xi,\eta}$ will be used for securing future communication between ξ and η. As an example of information used for key reconciliation, node ξ sends to node η the number $r = \langle K_{\xi,\eta} \rangle_{2^s}$, where $s = \lceil \log_2(4m+1) \rceil$. Node η can efficiently obtain the integer j such that $|j| \leq 2m$ and $K_{\xi,\eta} \equiv K_{\eta,\xi} + jN \mod 2^b$ by using that $jN \equiv K_{\xi,\eta} - K_{\eta,\xi} \equiv r - K_{\eta,\xi} \mod 2^s$. As N is odd, the latter equation allows for determination of j. As r reveals the s least significant bits of $K_{\xi,\eta}$, only the $b - s$ most significant bits $K_{\xi,\eta}$, that is, the number $\lfloor 2^{-s} K_{\xi,\eta} \rfloor$, should be used as key.

3.2 Implicit Certification and Verification of Credentials

Implicit certification and verification of credentials is further enabled on top of the basic HIMMO scheme. A node that wants to register with the system provides the TTP with its credentials, e.g., device type, manufacturing date, etc. The TTP, which can also add further information to the node's credentials such as a unique node identifier or the issue date of the keying material and its expiration date, obtains the node's identity as $\xi = H(\mathit{credentials})$, where H is a public hash function. When a first node with identity ξ wants to securely send a message M to a second node with identity η, the following steps are taken.

- Step 1: Node ξ computes a common key $K_{\xi,\eta}$ with node η, and uses $K_{\xi,\eta}$ to encrypt and authenticate its credentials and M, say $e = E_{K_{\xi,\eta}}(\mathit{credentials}|M)$.

- Step 2: Node ξ sends (ξ, e, r) to node η, where r is data helping node η to find $K_{\xi,\eta}$.
- Step 3: Node η receives (ξ', e', r'). Using r', it computes its common key $K_{\eta,\xi'}$ with ξ' to decrypt e' obtaining the message M and verifying the authenticity of the received message. Furthermore, it checks whether the $credentials'$ in e' correspond with ξ', that is, it validates if $\xi' = H(credentials')$.

This method not only allows for direct secure communication of message M, but also for implicit certification and verification of ξ's credentials because the key generating polynomial assigned to a node is linked to its credentials by means of H. If the output size of H is long enough, e.g., 256 bits, the input (i.e., the credentials) contains a unique node identifier, and if H is a secure one-way hash function, then it is infeasible for an attacker to find any other set of credentials leading to the same identity ξ. The fact that credential verification might be prone to birthday attacks motivates the choice for the relation between identifier and key sizes, namely, $B = 2b$. In this way, the scheme provides an equivalent security level for credential verification and key generation. The capability for credential verification enables e.g. the verification of the expiration date of the credentials (and the keying material) of a node, or verification of the access roles of the sender node ξ.

3.3 Enhancing Privacy by Using Multiple TTPs

Using multiple TTPs was introduced by Matsumoto and Imai [12] for KPS and can also be elegantly supported by HIMMO [9]. In this set-up, a number of TTPs provide a node with keying materials linked to the node's identifier during the keying material extraction phase. Upon reception, the device combines the different keying materials by adding the coefficients of the key generating polynomials modulo N. Without increasing the resource requirements at the nodes, this scheme enjoys two interesting properties. First, privacy is enhanced since a single TTP cannot eavesdrop the communication links. In fact, all TTPs should collude to monitor the communication links. Secondly, compromising a sub-set of TTPs does not break the overall system.

4 Implementation and Performance

HIMMO has been designed keeping in mind that we want to achieve very good performance. In this section, we explain how the key generation algorithm in Eq. 4 can be implemented in a very efficient way for the specific choice $N = 2^{B(\alpha+1)+b} - 1$, taking into account that the size of the identifiers (B bits) is small compared to the size of the coefficients of the polynomial G_ξ ($(\alpha+1)B + b$ bits). Algorithm 1 shows this optimized key generation algorithm that applies an approximation of the well-known Horner algorithm for evaluating polynomials in which the specific choice of N is taken into account. In the appendix, we motivate some steps of the algorithm and show that

$$\langle\langle\sum_{j=0}^{\alpha} G_{\xi,j}\eta^j\rangle_N\rangle_{2^b} \in \{key, \langle key+1\rangle_{2^b}\}. \tag{6}$$

Algorithm 1. Optimized key generation

```
 1: INPUT: $B$, $b$, $\alpha$, $\eta$, $G_{\xi,j}$ with $j \in \{0,\ldots,\alpha\}$
 2: OUTPUT: $key$
 3: $key \leftarrow \langle G_{\xi,\alpha}\rangle_{2^b}$
 4: $temp \leftarrow \lfloor \frac{G_{\xi,\alpha}}{2^b} \rfloor$
 5: for $j = \alpha - 1$ to $0$ do
 6:     $temp \leftarrow temp \times \eta + \lfloor \frac{G_{\xi,j}}{2^{(\alpha-1-j)B+b}} \rfloor$
 7:     $key \leftarrow \langle key \times \eta\rangle_{2^b} + \langle G_{\xi,j}\rangle_{2^b}$
 8:     $key \leftarrow \langle key + \lfloor \frac{temp}{2^{(j+2)B}} \rfloor\rangle_{2^b}$
 9:     $temp \leftarrow \lfloor \frac{\langle temp\rangle_{2^{(j+2)B}}}{2^B} \rfloor$
10: end for
11: return $key$
```

We note that part of the coefficients $G_{\xi,j}$ with $j \in \{0,\ldots,\alpha\}$ is not used in Algorithm 1. This allows for a further optimization in which only the required parts of the coefficients are stored, namely the b least significant bits and the jB most significant bits of each coefficient $G_{\xi,j}$.

Figure 1 provides a brief summary of the performance of the HIMMO scheme implemented in C and assembler on the 8-bit CPU ATMEGA128L. The first graph shows the key generation time for $\alpha = 26$ as a function of $b = B$. In the next two graphs we see – as a function of α and for $b = B = 128$ – the key generation time and the size of the key generating function.

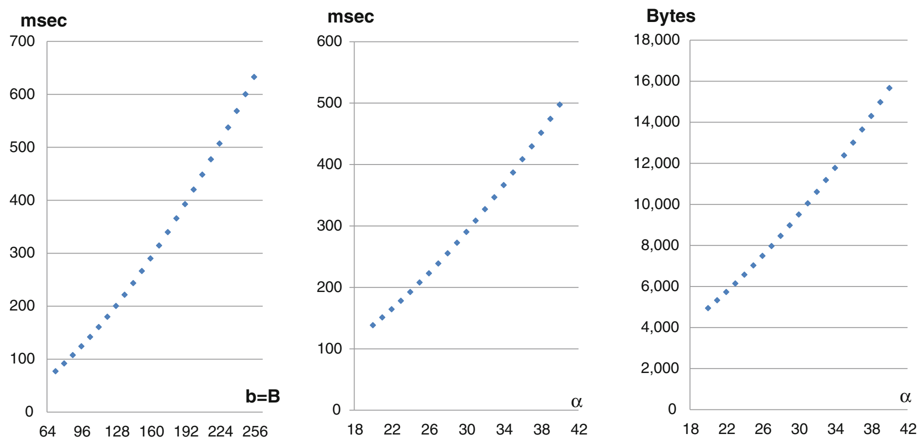

Fig. 1. HIMMO Performance: on the left, performance for $\alpha = 26$ as a function of $b = B$; in the middle and right, performance for $b = B = 128$ as a function of α.

Table 1. HIMMO performance and comparison with ECDH and ECDH + ECDSA.

	CPU time	Key material + code	RAM	Exchanged data	Security properties
ECDH [11]	3.97 s	16018 B	1774 B	480 B	Key agreement
ECDH + ECDSA [11]	11.9 s	35326 B	3284 B	704 B	Key agreement and credential verification
HIMMO	0.290 s	7560 B	1220 B	448 B	Key agreement and credential verification

Table 1 compares the performance HIMMO using security parameters $\alpha = 26$ and $2b = B = 160$ with that of ECDH and/or ECDSA for a security level of 80 bits. We illustrate a simple interaction scenario between two nodes: a first node ξ wants to send in a secure way information to η, and η wants to securely receive the message from ξ and verify its credentials. The first two protocols involve communicating before node ξ can send an encrypted message, whereas HIMMO allows node ξ to directly compute the key with η based on its identifier and send the encrypted message. Also, notice that ECDH only provides key agreement, to get key agreement and verification of credentials, it is needed to use also ECDSA, increasing the resource requirements. For this implementation, we use the ATMEGA128L processor running at 8 MHz since it is a typical resource-constrained device used in the IoT. Other less constrained devices are emerging featured by longer word size (32-bit) and slightly higher clock frequency. In Table 1, CPU refers to the overall computing needs, the memory refers to the amount of information that needs to be stored in flash, RAM is the RAM memory needs, exchanged data refers to the amount of data exchanged between ξ and η, and finally, the security properties illustrate the features of the security protocols.

5 (D)TLS-HIMMO

TLS and DTLS are two of the protocols to protect the Internet today, while DTLS is becoming the standard for the IoT. Existing (D)TLS operational modes have pros and cons. PSK is efficient but does not scale well. Raw-public key scales well but does not offer authentication and is prone to man-in-the-middle attacks. Certificate-based schemes are too expensive in some scenarios, in particular IoT related ones, and most of those schemes would also be broken with quantum computers. This motivates our research in a new (D)TLS cipher suite based on HIMMO that:

– has the low operational cost of DTLS-PSK,
– enables mutual authentication and credential verification as with certificate-based schemes,
– is scalable like public-key cryptography and infrastructure,
– is resilient to attacks using quantum computers.

To this end, we extend the DTLS-PSK mode, which is based on identities, without changing the standard so that it can work with HIMMO. The main difference from the usual PSK profile lies in using identities to generate a pairwise symmetric key and, then, deriving the session keys from the pairwise symmetric key. A TTP provisions keying material to client nodes and the server as shown in Eq. 2 during an initial setup (e.g. manufacture stage). HIMMO can be directly used in (D)TLS-PSK mode by exchanging HIMMO's identifiers in the *ClientKeyExchange* and *ServerKeyExchange* messages. Creation of a new profile to indicate DTLS-HIMMO (e.g. TLS_DTLS-HIMMO_WITH_AES_128_CCM_8) can also be considered, but requires standardization.

5.1 DTLS-HIMMO Configurations

The existing PSK profile, such as the one used in TLS_PSK_WITH_AES_128_CCM_8, involves the exchange of two fields, the *PSK-identity* and *PSK-identity-hint*, in the *ClientKeyExchange* and *ServerKeyExchange* messages respectively. Instead of sending a PSK identifier, we use these fields, which can be up to 128 bytes long [17], to exchange HIMMO information.

Table 2 illustrates these fields of information with exemplary lengths. First, we find an identifier/flag indicating the use of DTLS-HIMMO. Next, we find a DTLS-HIMMO message type. The third and fourth field refer to the number of TTPs and their identifiers, respectively. These are the TTPs associated with generating and distributing the key material of the client and server. These two fields are followed by an identifier field. Next, we optionally find the HIMMO credentials length as well as the credentials themselves. Finally, a field that contains the key reconciliation data is optionally present. The interpretation of the identifier field and the absence of presence of the optional fields is derived from the Message Type field.

This message format is used in the *PSK-identity-hint* and *PSK-identity* fields of the *ServerKeyExchange* and *ClientKeyExchange* messages. With these fields we can enable different ways of using HIMMO with DTLS-PSK. If only the HIMMO identifier is exchanged in the identifier field, then only mutual authentication is achieved between client and server. Alternatively, the client, or server, or both of them might exchange their credentials. The credentials could be any information that today is exchanged in regular digital certificates and, for IoT scenarios, information such as manufacturer, device type, date of manufacturing, etc. In this case, the exchange enables unilateral or mutual implicit credential verification of the parties. We note that in this case, the identifier field does not contain the HIMMO identifier but a unique random value that concatenated

Table 2. Exemplary format of the *PSK-identity-hint* and *PSK-identity* fields enabling DTLS-HIMMO; Length in Bytes; N = Number of TTPs; M = mandatory, O = optional

	HIMMO flag	Message type	Number of TTPs	TTP ID	Identifier	HIMMO credentials length	HIMMO credentials	Reconcilliation data
Length	2	1	1	N	B	1	$0 \ldots (122 - B - N)$	N
M/O	M	M	M	M	M	O	O	O

Table 3. Modes of operation of DTLS-HIMMO profile

		Client sends HIMMO's ID	Client sends HIMMO's credentials
Server sends HIMMO's ID	**Messages exchanged**	ClientKeyExchange: Client ID and Reconciliation data ServerKeyExchange: Server ID	ClientKeyExchange: Clients credentials and Reconciliation data ServerKeyExchange: Server ID
	Computations	Two HIMMO evaluations	Two HIMMO evaluations One hash evaluation
	Properties	Mutual authentication	Mutual authentication Verification of client's credentials
Server sends HIMMO's credentials	**Messages exchanged**	ClientKeyExchange: Client ID and Reconciliation data ServerKeyExchange: Servers credentials	ClientKeyExchange: Clients credentials and Reconciliation data ServerKeyExchange: Servers credentials
	Computations	Two HIMMO evaluations One hash evaluation	Two HIMMO evaluations Two hash evaluations
	Properties	Mutual authentication Verification of server's credentials	Mutual authentication Verification of the credentials of client and server

with the information in the *HIMMO credentials length* and *HIMMO credentials* is hashed to obtain the HIMMO identifier. The reason for this construction was explained in Sect. 3.2. Finally, we note that the reconciliation data is only exchanged in the *ClientKeyExchange* message since it is the server the one performing this operation.

These two different options give rise to four (two each for client and server) different combinations whose features are explained in Table 3.

5.2 (D)TLS-HIMMO Handshake

The HIMMO enabled PSK message exchanges comprises multiple steps:

– Step 1: The client sends a *ClientHello* message to the server indicating use of the PSK mode, such as the TLS_PSK_WITH_AES_128_CCM_8.
– Step 2: The usual *HelloVerifyRequest* message, with a cookie, is sent from the server to the client.
– Step 3: The client replies back with *ClientHello* along with the cookie.
– Step 4: The server replies with *ServerHello*, *ServerKeyExchange* and *ServerHelloDone*. The *PSK-identity-hint* of the *ServerKeyExchange* contains the DTLS-HIMMO fields as in the exemplary format shown in Table 2.
– Step 5: The client sends the *ClientKeyExchange* with the *PSK-identity* field containing the DTLS-HIMMO fields as shown in Table 2. It also sends the usual *ChangeCipherSpec* and *Finished* messages to the server.
– Step 6: The Server would send back the usual *ChangeCipherSpec* and *Finished* messages to the client.

The client computes the symmetric pairwise key as follows:

- Step 1: If the server sent its credentials, as indicated in the DTLS-HIMMO fields, compute *ID-Server* = H(*Server Identifier* || *Server HIMMO Credentials Length* || *Server HIMMO-credentials*). In case the server sent the HIMMO identifier then set *ID-Server* = *Server HIMMO-Identifier.*
- Step 2: If the client is also using credentials, compute *ID-Client* = *H*(*Client Identifier* ||*Client HIMMO Credentials Length*||*Client HIMMO-credentials*). Otherwise, set *ID-Client* = *Client HIMMO-Identifier.*
- Step 3: Compute the pairwise key $K_{\textit{ID-Client, ID-Server}}$ as shown in Eq. 4.

Similarly, the server, upon receipt of the *ClientKeyExchange* message computes the pairwise key as:

- Step 1: Depending upon whether the client sent its credentials or its HIMMO identifier, compute *ID-Client* as shown in the steps followed by the client before. In the same manner, depending upon whether the server uses credentials or its HIMMO identifier, compute *ID-Server.*
- Step 2: Compute the pairwise key $K_{\textit{ID-Server, ID-Client}}$ using the key reconciliation data sent by the client to arrive at the symmetric pairwise key.

Note that the respective key generating polynomials ($G_{\xi,k}$ in Eq. 2) in the devices would be configured with either the HIMMO identifier or the hash of the concatenation of the identifier, the length of the credentials and the credentials for its identity ξ, depending upon which mode of operation is used (see Table 3). Once the client and server have computed the pairwise key, it can be part of the input to the standard (D)TLS pseudo-random function used to derive the session keys for the DTLS session as is done with the PSK profile. The DTLS *Finished* message authenticates the handshake, and thus, authenticates both parties as having the correct keying material. If the communicating peer is using HIMMO credentials for the key exchange, then the successful completion of the *Finished* message implies that the credentials it provided are correct and, thus, authenticates the credentials of the peer.

5.3 Privacy Protection

Protecting the privacy of the communication links is fundamental. HIMMO and its extensions can be used to ensure the privacy of the involved communication parties.

A first aspect is the protection of the exchanged credentials that might contain some private information that should not be exposed to the other party, if not authenticated before, or to a passive eavesdropper. This can be achieved by the following simple extension of the DTLS handshake. The credentials are encrypted with the pairwise key shared with the other party. For instance, in the DTLS-HIMMO exchange, the client protects its credentials by encrypting them with the HIMMO key shared with the server and that is computed after the reception of the *ServerKeyExchange* message. Thus, the *ClientKeyExchange* could contain the Client's HIMMO identifier and encrypted HIMMO credentials. The server uses the HIMMO identifier to obtain the common pairwise key, and decrypts the client's credentials. Neither a fake server nor an attacker eavesdropping the communication is able to learn the client's credentials.

The usage of raw-public keys with out-of-band verification or of digital certificates requires some type of public-key infrastructure that allows validating the authenticity of the involved public-keys or installing the digital certificates in a secure way. A certification authority (CA), or a hierarchy of CAs, plays this role in today's public-key infrastructure (PKI). HIMMO relies on a TTP whose role is similar to that of a CA. Like a CA, the TTP is in charge of validating the identity of a joining node and securely distributing its key generating function. The difference is that a single TTP could be misused and the TTP (or anyone having access to it) could eavesdrop or alter the ongoing data exchanges between any pair of nodes in a passive way. As explained in Sect. 3.3, the usage of multiple TTPs avoids this situation, since each device then registers with several TTPs and combines the received key generating polynomials from each TTP. In this way, the generated keys between any pair of entities of the system depend on the information shared by all the involved TTPs. An active attacker that can compromise a TTP can act as follows: he sets up a new server and tries to make a client authenticate to that server by setting the number of TTPs in the *ServerKeyExchange* message equal to one. One way to protect against this attack is a policy that a client only authenticates if the number of TTPs in the *SeverKeyExchange* message is at least two.

5.4 TTP Infrastructure

The introduction of an infrastructure of TTPs for the DTLS-HIMMO profile would mean the creation of an alternative to today's PKI. As outlined above, each entity in the system would register with a number of TTPs and receive the corresponding key generating polynomials, each linked to the same or related credentials. Each entity would store this information either combined, as explained in Sect. 3.3, or independently. In this case, the TTP identifiers can be exchanged between client and server during a DTLS-HIMMO handshake. In a first step, the server provides in the *ServerKeyExchange* message the TTP identifiers from which it received its key generating polynomials. In a second step, the client answers with the common or chosen TTPs in the *ClientKeyExchange* message.

Such an infrastructure brings new challenges but also advantages. Today, if a CA is compromised, then it is not possible to easily recover since certificates are often not signed by more than one CA. On the other hand, if they are, recovering is feasible, but this rapidly increases the bandwidth and computational requirements since all those certificates need to be exchanged and verified. This is not the case for above TTP Infrastructure since the capture of a single TTP does not break the whole system and using more than a TTP (e.g., t) involves practically the same bandwidth and computational resources as when a single one is used.

5.5 Security Considerations of (D)TLS-HIMMO

In [9], it is shown that a collusion attack in HIMMO amounts to solving a close vector problem in a certain lattice, and that the minimum required number of nodes, and thus the lattice dimension, is $(\alpha + 1)(\alpha + 2)/2$. If α is large enough, $\alpha > 25$, an approximate solution of this close vector problem, using the default LLL [14] implementation of Sage [15], and Babai's nearest plane algorithm, fails to give a good result, while the lattice dimension becomes too large for exact methods, for which the running time and memory requirements grow exponentially in the lattice dimension. While it is quite likely that more elaborate approximate classical algorithms would give better results, thus increasing the minimum required value of α somewhat, currently no quantum algorithm exists that would speed up the approximate lattice methods, nor is it foreseen that the quantum speed-ups in the exact lattice algorithms, which use enumeration techniques, are sufficient to crack HIMMO for these values of α. Therefore, the scheme presented in this paper could be an interesting approach to ensure secure digital communications in the Internet in a post-quantum world.

6 Performance of DTLS-HIMMO and Comparison with Existing (D)TLS Alternatives

We have implemented the DTLS-HIMMO operation mode such that the client and server run on a Intel Core i5-3437U @ 1.90 GHz with Windows 7 Enterprise. The DTLS-HIMMO extension is carried out by using DTLSv1.2 in PSK mode as starting point as explained in Sect. 5. The HIMMO-based DTLS operation modes include: (i) HIMMO enabling mutual authentication, (ii) HIMMO enabling mutual authentication and server verification, and (iii) HIMMO enabling mutual authentication and client and server verification. We compare DTLS-HIMMO with (iv) DTLS in PSK mode, (v) DTLS certificates enabling server verification only and (vi) DTLS certificates with both server and client verification. Both modes are implemented using the ECDHE and ECDSA using the NIST secp256r1 curve for ECC computations. All of the analyzed DTLS operation modes rely on a 128-bit AES in CCM operation mode to secure the DTLS record layer.

Table 4 provides a qualitative comparison of the above DTLS modes of operation against their performance and security properties. Performance-wise we discuss the resource requirements on the client and server and the communication overhead. Security-wise we consider the capability of the handshakes for key agreement, authentication, information verification, and scalability.

Due to the identity-based nature of HIMMO, the verification of the client or server credentials only costs one additional hash computation. For this reason, the communication overhead can be kept at a very low level compared with certificates. For ECDHE + ECDSA, key agreement and verification of information requires several scalar ECC point multiplications, while HIMMO only requires a polynomial evaluation.

Table 4. Qualitative comparison of the HIMMO based PSK profile with other algorithms. All algorithms allow for key agreement

DTLS mode	Client CPU Needs	Server CPU Needs	Handshake size	Authentication	Information verification	Scalability
DTLS-HIMMO	HIMMO key generation	HIMMO key generation Key reconciliation	Low	Mutual	No	$G_{\xi}(x)$ installation
DTLS-HIMMO(SA) (Server authentication)	1 SHA-256 HIMMO key generation	HIMMO key generation Key reconciliation	Low	Mutual	Server authentication	$G_{\xi}(x)$ installation
DTLS-HIMMO (Mutual authentication)	1 SHA-256 HIMMO key generation	1 SHA-256 HIMMO key generation Key reconciliation	Low	Mutual	Server and Client	$G_{\xi}(x)$ installation
PSK	-	-	Low	Mutual	No	Installation of PSKs
ECDHE + ECDSA (Server authentication)	Three ECC point multiplications	One ECC point multiplication	High	Unilateral	Server verification	Root Certificate installation
ECDHE + ECDSA (Mutual authentication)	Three ECC point multiplications	Three ECC point multiplication	Higher	Mutual	Server and client verification	Root Certificate installation

This qualitative comparison is supported by the experimental results in which we have measured (i) the elapsed time, (ii) the amount of data exchanged, and (iii) the ratio between data exchanged and payload in three different scenarios for different DTLS modes of operation:

- the DTLS connection is established and 1 KB of data are exchanged,
- the DTLS connection is established and 10 KB of data are exchanged, and
- the DTLS connection is established and 100 KB of data are exchanged.

Figure 2 depicts the total amount of exchanged data for all the cipher suites. This includes the headers of the underlying protocols (UDP, IP, etc.) as well as the transfer of 1 KB of data. On the left side of Fig. 3 we see the required time to establish a secure connection and send the data for different cipher suites. On the right side of this figure we observe the ratio between the required bandwidth and the exchanged payload. In both figures, from top to bottom: (1) ECDH-ECDSA with mutual authentication, (2) ECDH-ECDSA with server authentication, (3) HIMMO with mutual verification of client's and server's credentials ($t = 5, B = 256, b = 32, \alpha = 17$), (4) HIMMO with mutual authentication ($t = 5, B = 32, b = 32, \alpha = 50$) and (5) PSK. We notice that DTLS-PSK is the fastest followed by DTLS-HIMMO without credential verification capabilities. DTLS-HIMMO with credential verification capabilities becomes slightly more expensive since B needs to be larger than the generated key in this case. We also observe that the value of the security parameter α does not heavily impact the performance of the scheme remaining around a factor 8 faster than the ECC alternative. We note that in this experiment HIMMO is configured to generate a key 128 bit long by combining five ($t = 5$) instances in parallel. For the cases of mutual authentication and mutual credential verification we use HIMMO parameters ($B = 32, b = 32, \alpha = 50$) and ($B = 256, b = 32, \alpha = 17$) respectively. This implies that an attacker has to deal with lattices of dimensions as high as 1377 and 1368, respectively, for the HI problem [8]. It is also worth noting that in all cases the cryptographic operations involved in the transfer of data are negligible

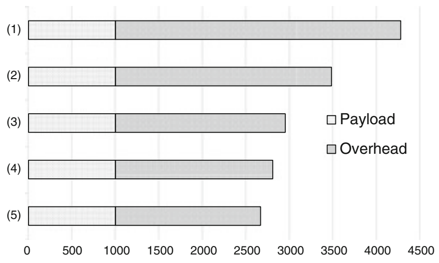

Fig. 2. Exchanged data.

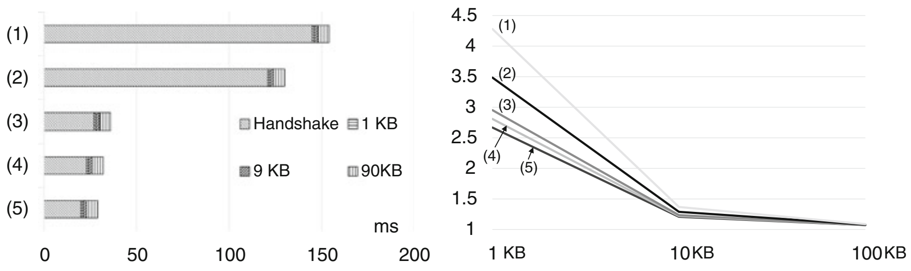

Fig. 3. DTLS connection time (left) and ratio between total exchanged data and payload (right).

compared with the DTLS handshake. Figure 2, right side, shows that the usage of schemes relying on long keys might not be the best solution for use cases in which little payload needs to be exchanged.

These figures together with Fig. 1 show several advantages of HIMMO compared with other alternatives. The first one is that IoT applications that involve the exchange of little data, frequently under 10 KB, can benefit from HIMMO since it offers a better ratio between the amount of transmitted payload and the overall amount of transmitted data. This is due to HIMMO's identity based nature that does not require the exchange of public-keys or long certificates. As a result, the underlying constrained networks are less overloaded, thus enabling IoT applications with less costs to network operators. The second advantage is that same back-end can handle many more clients with the same resources. This prevents potential DoS attacks and decreases again the price to enable those applications. Finally, we remark that Fig. 3 shows the DTLS connection time between two powerful devices. In a real world IoT scenario one of those devices will have much lower capabilities. However, HIMMO can be still implemented in a very efficient way as illustrated in Fig. 1.

7 Conclusions

The HIMMO scheme is the first Key Pre-distribution scheme that is simultaneously efficient and secure (in terms of collusion resistance). Specific choices of the HIMMO parameters enable very efficient implementations that, combined with the implicit credential certification and verification, improve the performance of related public-key schemes by one order of magnitude. HIMMO can be embedded in TLS and DTLS, the security protocols used to secure the Internet, without requiring any changes in the standards, but offering a significantly improved performance security trade-off. In fact, the DTLS-PSK mode can be extended with HIMMO to achieve functionality that today is only possible with public-key cryptography and a public-key infrastructure, but at the speed and memory requirements of a symmetric-key handshake. The DTLS-HIMMO handshake offers mutual authentication of client and server, implicit verification of their credentials costing a single hash computation, client's privacy-protection by sending its credentials in encrypted format, and support of multiple TTPs.

We finally remark that HIMMO is post-quantum secure as known attacks involve solving a close vector problem in a lattice for which currently no quantum algorithm exists that would speed up the approximate lattice methods, nor is it foreseen that the quantum speed-ups in the exact lattice algorithms, which use enumeration techniques, are sufficient to crack HIMMO.

Appendix: Proof of Correctness of Optimized Algorithm

Algorithm 1 is an approximation to Horner's algorithm for polynomial evaluation modulo N, taking into account that N is of the special form $N = 2^{(\alpha+1)B+b}$ and that the argument η is small. In this appendix, we motivate some of the steps in Algorithm 1, and prove (6) which states that the output of the algorithm nearly is the wanted key.

Each intermediate value in Horner's algorithm for computing $\langle \sum_{j=0}^{\alpha} G_{\xi,j}\eta^j \rangle_N$ is obtained as

$$\langle temp_j \rangle_N = \langle temp_{j+1} \times \eta + G_{\xi,j} \rangle_N$$

for $j = \alpha - 1, \ldots, 0$. As $0 \leq \eta < 2^B$, we can write $temp_{j+1} \times \eta + G_{\xi,j} = temp_j = temp_j^H \times 2^{(\alpha+1)B+b} + temp_j^L$, where $temp_j^H$ and $temp_j^L$ are b and $(\alpha+1)B+b$ bits long, respectively. As $N = 2^{(\alpha+1)B+b} - 1$, we thus have that $\langle temp_j \rangle_N = \langle temp_J^H + temp_j^L \rangle_N \approx temp_j^H + temp_j^L$. This is only an approximation because there might be a carry in the addition of $temp_j^H$ and $temp_j^L$, requiring a second reduction. We will show that this second reduction is needed at most once during the calculation, and ignoring it leads to a difference of one (mod 2^b) between the wanted key and the value returned by the algorithm, so that (6) is satisfied. The modular reduction happens when the value of *key* is updated with the contribution of the MSB stored in *temp* after being shifted $(j+2)B$ bits and added to *key* (Line 8).

We now state and prove the main property of Algorithm 1. Let b, B, α be positive integers and let $N := 2^{(\alpha+1)B+b} - 1$. For $0 \le i \le \alpha$, let $0 \le G_i \le N-1$, and let $0 \le \eta \le 2^B - 1$. We are interested in obtaining the key K, defined as

$$K := \langle \langle \sum_{i=0}^{\alpha} G_i \eta^i \rangle_N \rangle_{2^b}. \tag{7}$$

For $0 \le i \le \alpha - 1$, we write

$$G_i = \gamma_i 2^{(\alpha-i-1)B+b} + \delta_i \text{ with } 0 \le \delta_i \le 2^{(\alpha-i-1)B+b} - 1. \tag{8}$$

We rewrite Algortihm 1, where we added indices to the variables that will be useful in the analysis the algorithm:

```
$k_\alpha := \langle G_\alpha \rangle_{2^b}$; $\tau_\alpha := \lfloor \frac{G_\alpha}{2^b} \rfloor$;
for $j := \alpha - 1$ downto $0$ do
begin $\sigma_j := \tau_{j+1} \times \eta + \gamma_j$;
      $k_j := \langle k_{j+1} \times \eta + \langle G_j \rangle_{2^b} + \lfloor \frac{\sigma_j}{2^{(j+2)B}} \rfloor \rangle_{2^b}$;
      $\tau_j := \lfloor \frac{\langle \sigma_j \rangle_{2^{(j+2)B}}}{2^B} \rfloor$
end;
key:= $k_0$
```

Theorem 1. *If $\alpha < 2^B$, then either $K = key$ or $K = \langle key + 1 \rangle_{2^b}$.*

For proving the above theorem, we define $\Lambda_\alpha, \Lambda_{\alpha-1}, \dots, \Lambda_0$ as

$$\Lambda_\alpha := G_\alpha \text{ and for } 0 \le j \le \alpha - 1, \Lambda_j := \eta \Lambda_{j+1} + G_j - \lfloor \frac{\sigma_j}{2^{(j+2)B}} \rfloor N. \tag{9}$$

By induction on j, it is easy to see that for $0 \le j \le \alpha$, $\Lambda_j \equiv \sum_{i=j}^{\alpha} G_i \eta^{i-j} \bmod N$. Note that $\sum_{i=j}^{\alpha} G_i \eta^{i-j}$ is the j-th iterate of the evaluation of $\sum_{i=0}^{\alpha} G_i \eta^i$ using Horner's algorithm.
We will show below (Proposition 2) that for each j,

$$0 \le \Lambda_j - \tau_j 2^{(\alpha-j)B+b} \le (\alpha - j + 1) 2^{(\alpha-j)B+b}.$$

As a consequence, if $\alpha < 2^B$, then $0 \le \Lambda_0 - \tau_0 2^{\alpha B+b} < N$. The algorithm implies that $0 \le \tau_0 \le 2^B - 1$, and so $0 \le \tau_0 \le \Lambda_0 < N + 2^B - 1$. As $\sum_{j=0}^{\alpha} G_j \eta^j \equiv \Lambda_0 \bmod N$, we conclude that $\langle \sum_{j=0}^{\alpha} G_j \eta^j \rangle_N = \langle \Lambda_0 \rangle_N \in \{\Lambda_0, \Lambda_0 - N\}$, and so

$$K \in \{\langle \Lambda_0 \rangle_{2^b}, \langle \Lambda_0 + 1 \rangle_{2^b}\}. \tag{10}$$

In Proposition 3, we show that $\Lambda_j \equiv k_j$ for $0 \le j \le \alpha$. Combining this result with (10) proves the theorem.

For $0 \le j \le \alpha$, we define

$$r_j := \Lambda_j - 2^{(\alpha-j)B+b} \tau_j.$$

Proposition 1. *For* $0 \leq j \leq \alpha - 1$*, we have that* $r_j = 2^{(\alpha-j-1)B+b}\langle\sigma_j\rangle_{2^B} + \eta r_{j+1} + \delta_j + \lfloor\frac{\sigma_j}{2^{(j+2)B}}\rfloor$.

Proof. Let $0 \leq j \leq \alpha - 1$. From the definitions of $\Lambda_j, \Lambda_{j+1}, r_j$ r_{j+1} and σ_j we readily find that

$$r_j = 2^{(\alpha-1-j)B+b}(\sigma_j - 2^B\tau_j) + \eta r_{j+1} + \eta\delta_j - \lfloor\frac{\sigma_j}{2^{(j+2)B}}\rfloor N.$$

Writing $\sigma_j = \lfloor\frac{\sigma_j}{2^{(j+2)B}}\rfloor 2^{(j+2)B} + \langle\sigma_j\rangle_{2^{(j+2)B}}$, and using that $N = 2^{(\alpha+1)B+b} - 1$, we obtain that

$$r_j = 2^{(\alpha-1-j)B+b}(\langle\sigma_j\rangle_{2^{(j+2)B}} - 2^B\tau_j) + \lfloor\frac{\sigma_j}{2^{(j+2)B}}\rfloor + \eta r_{j+1} + \eta\delta_j.$$

The proposition now follows from observing that

$$\langle\sigma_j\rangle_{2^{(j+2)B}} = 2^B\lfloor\frac{\langle\sigma_j\rangle_{2^{(j+2)B}}}{2^B}\rfloor + \langle\langle\sigma_j\rangle_{2^{(j+2)B}}\rangle_{2^B} = 2^B\tau_j + \langle\sigma_j\rangle_{2^B}.$$

□

Proposition 2. *For* $0 \leq j \leq \alpha$ *we have that* $r_j \leq (\alpha - j + 1)2^{(\alpha-j)B+b} - 1$.

Proof. By induction on j. As $r_\alpha = \langle G_\alpha\rangle_{2^b} \leq 2^b - 1$, the proposition is true for $j = \alpha$.
Now let $0 \leq j \leq \alpha - 1$. The algorithm immediately implies that $\tau_{j+1} \leq 2^{(j+2)B} - 1$ (make distinctions for $j = \alpha - 1$ and $j < \alpha - 1$ for showing this). Moreover,

$$\gamma_j = \lfloor\frac{G_j}{2^{(\alpha-j-1)B}}\rfloor \leq \frac{G_j}{2^{(\alpha-j-1)B+b}} \leq \frac{N-1}{2^{(\alpha-j-1)B+b}} \leq 2^{(j+2)B} - 1.$$

We conclude that

$$\sigma_j = \tau_{j+1}\eta + \gamma_j < 2^{(j+2)B}(\eta + 1) < 2^{(j+3)B}, \text{ and so}$$

$$\lfloor\frac{\sigma_j}{2^{(j+2)B}}\rfloor \leq 2^B - 1. \tag{11}$$

According to (8), we have that $\delta_j \leq 2^{(\alpha-1-j)B+b} - 1$, and clearly $\langle\sigma_j\rangle_{2^B} \leq 2^B - 1$. Combining these inequalities with (11) and Proposition 2, we infer that

$$\begin{aligned} r_j &\leq 2^{(\alpha-j-1)B+b}(2^B - 1) + (2^{(\alpha-1-j)B+b} - 1) \quad + \eta r_{j+1} + (2^B - 1) \\ &= 2^{(\alpha-j)B+b} + \eta r_{j+1} + 2^B - 2 < 2^{(\alpha-j)B+b} + 2^B(r_{j+1} + 1). \end{aligned}$$

According to the induction hypothesis, $r_{j+1} \leq (\alpha - j)2^{(\alpha-j-1)B+b} - 1$, and so

$$r_j \leq (\alpha - j + 1)2^{(\alpha-j)B+b} - 1.$$

□

Proposition 3. *For* $0 \leq j \leq \alpha$*, we have that* $k_j = \langle\Lambda_j\rangle_{2^b}$.

Proof. By induction on j. The proposition is true for $j = \alpha$.
Now let $0 \leq j \leq \alpha - 1$. The definition of Λ_j implies that

$$\Lambda_j = \eta\Lambda_{j+1} + G_j - \lfloor\frac{\sigma_j}{2^{(j+2)B}}\rfloor(2^{(\alpha+1)B+b} - 1) \equiv \eta\langle\Lambda_{j+1}\rangle_{2^b} + \langle G_j\rangle_{2^b} + \lfloor\frac{\sigma_j}{2^{(j+2)B}}\rfloor \pmod{2^b}.$$

As $k_{j+1} \equiv \Lambda_{j+1} \pmod{2^b}$, the definition of k_j implies the proposition. □

References

1. HP report: Internet of Things Research Study. www.fortifyprotect.com. Accessed 21 August 2014
2. TLS Ciphersuites. https://www.thesprawl.org/research/tls-and-ssl-cipher-suites
3. NIST workshop on cybersecurity in a post-quantum world (2015). http://www.nist.gov/itl/csd/ct/post-quantum-crypto-workshop-2015.cfm
4. Blundo, C., de Santis, A., Herzberg, A., Kutten, S., Vaccaro, U., Yung, M.: Perfectly secure key distribution for dynamic conferences. Inf. Comput. **146**, 1–23 (1998)
5. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176
6. Eronen, P., Tschofenig, H.: Pre-Shared Key Ciphersuites for Transport Layer Security (TLS). RFC 4279 (Proposed Standard), December 2005
7. García-Morchón, O., Gómez-Pérez, D., Gutiérrez, J., Rietman, R., Tolhuizen, L.: The MMO problem. In: Proceedings of ISSAC 2014, pp. 186–193. ACM (2014)
8. García-Morchón, O., Rietman, R., Shparlinski, I.E., Tolhuizen, L.: Interpolation and approximation of polynomials in finite fields over a short interval from noisy values. Exp. Math. **23**, 241–260 (2014)
9. García-Morchón, O., Gómez-Pérez, D., Gutiérrez, J., Rietman, R., Schoenmakers, B., Tolhuizen, L.: HIMMO - A Lightweight, Fully Colluison Resistant Key-Predistribution Scheme. Cryptology ePrint Archive, Report 2014/698 (2014). http://eprint.iacr.org/
10. Garcia-Morchón, O., Tolhuizen, L., Gomez, D., Gutierrez, J.: Towards full collusion resistant ID-based establishment of pairwise keys. In: Extended abstracts of the Third Workshop on Mathematical Cryptology (WMC 2012) and the Third International Conference on Symbolic Computation and Cryptography (SCC 2012), pp. 30–36 (2012)
11. Liu, A., Ning, P.: Tinyecc: a configurable library for elliptic curve cryptography in wireless sensor networks. In: Proceedings of the 7th International Conference on Information Processing in Sensor Networks, IPSN 2008, pp. 245–256. IEEE Computer Society, Washington, DC (2008)
12. Matsumoto, T., Imai, H.: On the key predistribution system: a practical solution to the key distribution problem. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 185–193. Springer, Heidelberg (1988)
13. McGrew, D., Bailey, D.: AES-CCM Cipher Suites for Transport Layer Security (TLS). RFC 6655 (Proposed Standard), July 2012
14. Nguyen, P.Q., Vallée, B. (eds.): The LLL Algorithm - Survey and Applications. Springer, Heidelberg (2010)
15. Sage. http://www.sagemath.org
16. Shelby, Z., Hartke, K., Bormann, C.: The Constrained Application Protocol (CoAP). RFC 7252 (Proposed Standard), June 2014
17. Tschofenig, H.: A Datagram Transport Layer Security (DTLS) 1.2 Profile for the Internet of Things, August 2014