DTLS-HIMMO: Efficiently Securing a Post-Quantum World with a Fully-Collusion Resistant KPS

Oscar Garcia-Morchon, Ronald Rietman, Sahil Sharma, Ludo Tolhuizen and Jose Luis Torre-Arce, Philips Group Innovation, Research, Eindhoven, The Netherlands;
e-mail: {oscar.garcia, ronald.rietman, sahil.sharma, ludo.tolhuizen, jose.luis.torre.arce}@philips.com

Abstract

The future development of quantum-computers could turn many key agreement algorithms used in the Internet today fully insecure, endangering many applications such as online banking, e-commerce, e-health, etc. At the same time, the Internet is further evolving to enable the Internet of Things (IoT) in which billions of devices deployed in critical applications like healthcare, smart cities and smart energy are being connected to the Internet. The IoT not only requires strong and quantum-secure security, as current Internet applications, but also efficient operation. The recently introduced HIMMO scheme enables lightweight identity-based key sharing and verification of credentials in a non-interactive way. The collusion resistance properties of HIMMO enable direct secure communication between any pair of Internet-connected devices. The facts that attacking HIMMO requires lattice techniques and that it is extremely lightweight make HIMMO an ideal lightweight approach for key agreement and information verification in a post-quantum world.

Building on the HIMMO scheme, this paper firstly shows how HIMMO can be efficiently implemented even in resource-constrained devices enabling combined key agreement and credential verification one order of magnitude more efficiently than using ECDH-ECDSA. while being quantum secure. 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, however, 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 DTLS-Certificate security suite while being quantum secure and exhibiting the overhead of symmetric-key primitives.

1 Introduction

The advent of quantum computers can mean that most of the algorithms used for key agreement or information verification are not secure any more. This would have severe consequences for the Internet and all the applications, e.g., e-banking, e-commerce, or e-health, that depend on its security protocols TLS and IPSec. At the same time, 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 not only IoT needs strong and post-quantum secure solutions, as today's Internet, but also efficient approaches to secure the data between smart devices, and between smart devices and the Internet.

Several schemes have been proposed in the last years and are believed to be quantum secure since they rely on methods for which quantum computers do not provide any significant advantage. Example of these schemes are NTRU, public-key systems based on the Learning With Errors problem. The limitation of these schemes is that they very frequently involve higher computational costs, longer keys or both. NTRU has excellent performance, but requires rather long keys [14]. Schemes based on the LWE problem, e.g. [1], perform worse than existing public-key solutions.

The Transport Layer Security (TLS)[2] 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 OneM2M, OMA LWM2M, etc. However, with the advent of quantum computers most of the cipher suites of (D)TLS will become insecure. Furthermore, already today, it is very frequently discussed that DTLS and its cipher suites are too heavy for many IoT use cases. Thus, there is a need for a (D)TLS cipher suite that is post-quantum secure, efficient, scalable, and simple to use.

It is estimated that currently 70 % of the IoT devices have security risks and are often poorly managed [1]. Having such a (D)TLS cipher suite would help to address these issues in an efficient post-quantum secure way. In some cases, IoT scenarios are not secure due to the resource limitations (e.g., memory or energy) of end devices that may not be able to support 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, bandwidth consumption plays a role since the devices are managed over a cellular connection and each extra byte costs money. The availability of quantum computers would make the situation even worse since most existing cipher-suites would be broken and most quantum resistant alternatives are relatively expensive resource-wise.

The HIMMO scheme [6], [7] is a fully-collusion resistant key pre-distribution scheme that enables lightweight identity-based key sharing and verification of credentials between devices in a single message,which is ideal for real time IoT interactions. With HIMMO, a device can directly generate a common key with another device based on its identity in a very efficient way. We believe that HIMMO is a good candidate in a post-quantum world since existing attacks require lattice techniques that are not known to be efficiently implementable in quantum computers. Finally and very importantly, HIMMO is extremely efficient so that it can enable secure communication links even in IoT scenarios.

This paper builds on the HIMMO scheme by showing how HIMMO can be efficiently implemented leading to an operation that is around one order of magnitude faster than public-key based solutions based on ECDH and ECDSA. 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 quantum-secure 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 while being post-quantum secure.

The rest of this paper is organized as follows. Section 2 describes the features of IoT scenarios, security needs, and reviews important IoT security standards. Section 3 reviews the HIMMO scheme and extensions. Section 4 discusses why HIMMO is a good candidate in a post-quantum world. Section 5 presents an efficient algorithm for key agreement and performance results. Section 6 introduces the (D)TLS-HIMMO operation mode. In Section 7, we compare DTLS-HIMMO with existing (D)TLS alternatives. Section 8 concludes this paper and discusses future work.

2 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 cipher-suite, 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, to 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

1. HP report. Internet of Things Research Study, www.fortifyprotect.com HP _IoT _Research _Study.pdf, retrieved on August 21 2014.

communicate end-to-end with a back-end server, reporting information such as energy consumption, maintenance data, etc by means of protocols such as OneM2M or 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[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 are also popular due to their smaller footprint and simple API [3] supporting more that 70 cipher-suites including different modes for the key agreement such as *ECDH, ECDHE, ECDSA, ECDSA, RSA, PSK*, several hash functions used in the generation of a message authentication code, e.g., *SHA, MD5, SHA256, SHA384* as well as several encryption algorithms, e.g., *RC4, 3DES, AES128, AES256, Camellia128, Camellia256* that can be configured in several block cipher modes such as *CCM, GCM*.

If quantum computers were introduced today, all the cipher suites available for key agreement based on ECC and RSA would become insecure since Shor's algorithm [17] (or modification of it) allow for efficient integer factorization on a quantum computer [12]. ECC algorithms would be potentially easier to attack than RSA since computers with a lower number of qubits are required in practice. For instance, *"A 160 bit elliptic curve cryptographic key could be broken on a quantum computer using around 1000 qubits while factoring the security-wise equivalent 1024 bit RSA modulus would require about 2000 qubits."* [12]. On the other hand, it is also likely that once a quantum computer of 1000 qubits is available, it is only a matter of a few months until the number of available qubits doubles.

The advent of the Internet of Things puts further pressure on available schemes since these smart devices that rely on DTLS to communicate with each other and with back-end systems have limited resources from a point of view of CPU, energy and bandwidth. This requires efficient cryptographic schemes due to several reasons. First, resource-constrained devices have a relatively constrained CPU and have to run on batteries for many years: the usage of computationally hungry solutions require more powerful devices and decrease the device lifetime. Second, back-end systems will have to manage millions of devices: the usage of expensive cryptographic solutions means that back-end systems will require many more resources. Third, communication often happens over resource-constrained networks such as IEEE 802.15.4/6LoWPAN that are lossy, have a low-data rate (250 kbits/sec) and have a limited message size (127 B): long keys or certificates are not recommended since they need to be fragmented leading to a considerable decrease in performance. Fourth, data communication often happens over cellular connections that are not free of charge: if long keys or certificates are involved, then the cost of exchanging a few bytes of information can easily increase several times due to this additional overhead.

2.1 DTLS-PSK

The Pre Shared Key (PSK) is an authentication and key exchange algorithm used in cipher suites, both in TLS [2] and DTLS. Although not in common use on the Internet, (D)TLS- PSK is widely employed by devices that are part of the Internet of Things since it has very low resource needs. We will also use this mode to enable DTLS-HIMMO. The ciphersuite TLS_PSK _WITH _AES _128 _CCM _8 [10], for instance, uses PSK as the authentication and key exchange algorithm.

Figure 1 illustrates PSK based authentication [3], as applied to the DTLS handshake. Since 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 [18], so the *ServerKeyExchange*

2. For instance, see https://www.thesprawl.org/research/tls-and-ssl-cipher-suites/
3. http://www.yassl.com/yaSSL/Products-cyassl.html

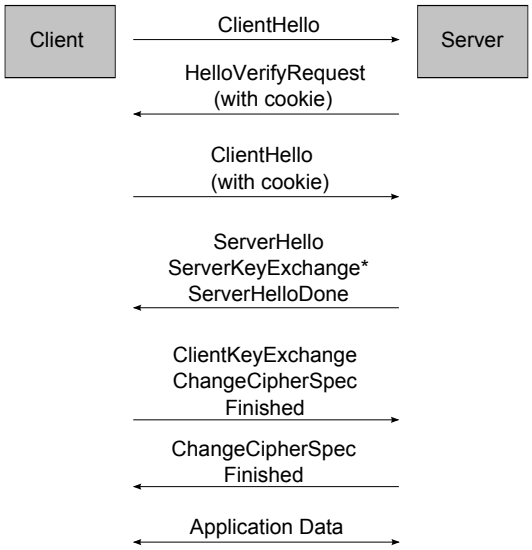

Figure 1. A DTLS PSK exchange (with cookies)

is optional (marked with *). 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 [2]. The cookie exchange is used to prevent denial of service attacks on the server. The Constrained Application Protocol (CoAP) [15] mandates the use of TLS_PSK _WITH _AES _128 _CCM _8 for the use with shared secrets [18].

3 HIMMO AND HIMMO EXTENSIONS

The concept of Key Pre-Distribution Schemes (KPS) was introduced by Matsumoto and Imai in 1987 [9]. However, there was no known KPS that is both efficient and not prone to efficient attacks of multiple colluding (or compromised) nodes. The HIMMO scheme solves 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. Furthermore, we describe two protocol extensions of the HIMMO scheme as introduced in [6], [7]. The underlying security principles on which HIMMO relies have been analyzed in [4] and [5].

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 $x \equiv y \bmod M$.

3.1 HIMMO operation

Like any KPS, HIMMO requires a trusted third party (TTP), and three phases can be distinguished in its operation [9].

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

$$R^{(i)}(x,y) = \sum_{j=0}^{\alpha} \sum_{k=0}^{\alpha} R_{j,k}^{(i)} x^j y^k$$

$$\text{with } 0 \leq R_{j,k}^{(i)} = R_{k,j}^{(i)} \leq q_i - 1.$$

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:

$$G_\xi(y) = \sum_{k=0}^{\alpha} G_{\xi,k} y^k \tag{1}$$

where

$$G_{\xi,k} = \langle \sum_{i=1}^{m} \langle \sum_{j=0}^{\alpha} R_{j,k}^{(i)} \xi^j \rangle_{q_i} \rangle_N. \tag{2}$$

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

$$K_{\xi,\eta} = \left\langle \langle G_\xi(\eta) \rangle_N \right\rangle_{2^b} \tag{3}$$

It can be shown that $K_{\xi,\eta}$ and $K_{\eta,\xi}$ need not be equal. However, as shown in Theorem 1 in [6], for all identifiers ξ and η with $0 \leq \xi, \eta \leq 2^B$,

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

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 $K_{\xi,\eta} \equiv K_{\eta,\xi} + jN \bmod 2^b$ by using that $jN \equiv K_{\xi,\eta} - K_{\eta,\xi} \equiv r - K_{\eta,\xi} \bmod 2^s$. 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 $\xi = H(credentials)$, 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 message M, say $e = E_{K_{\xi,\eta}}(\textit{credentials}|M)$.
- Step 2: Node ξ sends (ξ, e) to node η.
- Step 3: Node η receives (ξ', e'). 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 $\xi' = H(\textit{credentials'})$.

This method not only allows not 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, andif 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 [9] for KPS and can also be elegantly supported by HIMMO [6]. In this scheme, 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. Key generation is performed as usual. This scheme enjoys two interesting properties without increasing the resource requirements of the nodes. 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 HIMMO IN A POST-QUANTUM WORLD

In a collusion attack on the HIMMO scheme, multiple nodes collaborate in emulating the key generation process (3) of a node under attack, using their own pairwise keys with the node under attack as input. In [6], it is shown that this attack 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, $\alpha > 20$, an approximate solution of this close vector problem, using the default LLL [11] implementation of Sage [13], 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.

5 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 Equation 3 can be implemented in a very efficient way.

As we see in Equation 3, the key generation consists of the evaluation of a polynomial module N and taking the b LSBs. A good choice for N is $2^{B(\alpha+1)+b}-1$ because this simplifies the implementation of modular reductions on the devices. In Algorithm 1 we show the key generation algorithm whose underlying method is the well-known Horner's Rule. Each intermediate value is computed as follows

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

for $j = \alpha - 1, \ldots, 0$. To perform the modular reduction we take advantage of N's specific form, $2^{B(\alpha+1)+b} - 1$ and the small size of η. Thus, $\langle temp_j \rangle_N = \langle temp_j^H \times 2^{(\alpha+1)B+b} + temp_j^L \rangle_N \approx temp_j^H + temp_j^L$ where $temp_j^H$ and $temp_j^L$ are b and $(\alpha+1)B+b$ bits long, respectively. That this is an approximation is because there might be a carry in the addition of $temp_j^H$ and $temp_j^L$, requiring a second reduction. However, as shown in the appendix, this second reduction is

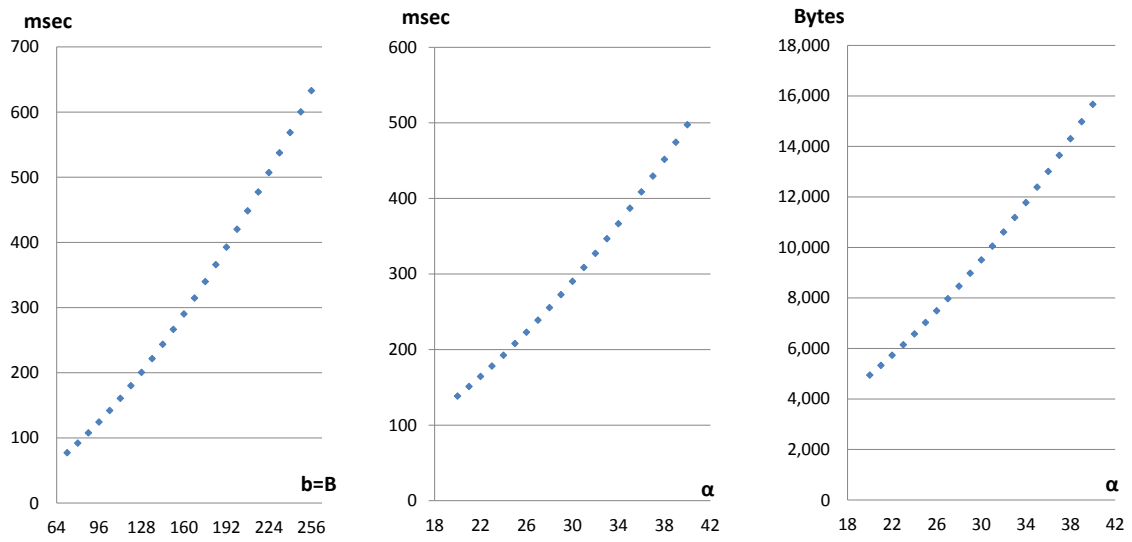

Figure 2. HIMMO Performance: on the left, performance for $\alpha = 26$ as a function of $b = B$; in the middle and right, performance for $b = B = 128$ as a function of α.

needed at most once during the calculation, and ignoring it leads to a difference of one $(\text{mod } 2^b)$ between the wanted key and the value returned by the algorithm, so that

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

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).

Algorithm 1 Optimized key generation

1: **INPUT:** B, b, α, η, $G_{\xi,j}$ 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: $\quad temp \leftarrow temp \times \eta + \lfloor \frac{G_{\xi,j}}{2^{(\alpha-1-j)B+b}} \rfloor$
7: $\quad key \leftarrow \langle key \times \eta \rangle_{2^b} + \langle G_{\xi,j} \rangle_{2^b}$
8: $\quad key \leftarrow \langle key + \lfloor \frac{temp}{2^{(j+2)B}} \rfloor \rangle_{2^b}$
9: $\quad temp \leftarrow \lfloor \frac{\langle temp \rangle_{2^{(j+2)B}}}{2^B} \rfloor$
10: **end for**
11: **return** key

From Algorithm 1 it is also clear that part of the coefficients $G_{\xi,j}$ with $j \in \{0, \ldots, \alpha\}$ are not used in the key generation process. This is because of the smaller size of the HIMMO identifiers (B bits) compared with the relatively long coefficients ($(\alpha+1)B$ bits). 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 2 provides a brief summary of the performance of the HIMMO scheme 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 figures, we see – as a function of α and for $b = B = 128$ – the key generation time and the size of the key generating function.

We further include a comparison table (Table 1) to illustrate the performance advantages of HIMMO compared with ECDH and ECDSA when implementing a simple interaction 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. The results are based on an implementation on the ATMEGA128L running at 8 MHz and illustrate the performance when this protocol is implemented with ECDH only, ECDH and ECDSA for a security level of 80 bits and HIMMO using security parameters $\alpha = 26$ and $2b = B = 160$. 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, round trips are the number of interactions between both nodes, and finally, the security properties illustrate the features of the security protocols.

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

	CPU time	Key material + code	RAM	Exchanged data	Security properties
ECDH [8]	3.97 s	16018 B	1774 B	480 B	Key agreement
ECDH +ECDSA [8]	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

6 (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 and quantum secure, but it does not scale. Raw-public key scales well but does not offer authentication, is prone to man-in-the-middle attacks, and most of existing schemes would be broken with quantum computers. Certificate-based schemes are too expensive in some scenarios, in particular Internet of Things related, 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:

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

To this end, we take the DTLS-PSK mode that is based on identities, and we extend – without need of 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 Equation 1 during an initial setup (eg. manufacture stage). HIMMO can be directly used in (D)TLS-PSK mode by exchanging HIMMO's identifiers in the *ClientKeyExchange* and *ServerKeyExchange* messages.[4]

6.1 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

4. Creation of a new profile to indicate DTLS-HIMMO (eg. TLS_DTLS-HIMMO _WITH _AES _128 _CCM _8) can also be considered, but requires standardization.

ServerKeyExchange messages respectively. Instead of sending a PSK identifier, we use these fields, which can be up to 128 bytes long [18], 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 to indicate which properties are enabled by HIMMO. The third and fourth field refer to the number of TTPs as well as their identifiers. 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. Next, we optionally find the HIMMO credentials length as well as the credentials themselves. Finally, a field that contains the key reconciliation data is present.

Table 2
Exemplary format of the PSK-identity-hint and PSK-identity fields enabling DTLS-HIMMO

	HIMMO flag	Message Type	Number of TTPs	TTP ID	Identifier	HIMMO Credentials length	HIMMO Credentials	Reconci-lliation data
Length(Bytes)	2	1	1	1/ # TTP	B	1	$0 \dots (122 - B)$	1
Mandatory (M)/ Optional (O)	M	M	M	M	M	O	O	O

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 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 Section 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 gives rise to four (two each for client and server) different combinations. These combinations, required computations and properties are shown in Table 3.

6.2 (D)TLS-HIMMO Handshake

The HIMMO enabled PSK message exchanges are shown in Figure 6.2, with the respective steps explained below:

- 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:

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 in total	Two HIMMO evaluations in total 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 in total One hash evaluation	Two HIMMO evaluations in total Two hash evaluations
	Properties	
	Mutual authentication Verification of server's credentials	Mutual authentication Verification of the credentials of client and server

- Step 1: If the server sent its credentials, as indicated in the DTLS-HIMMO fields, compute

$$\begin{aligned} \textit{ID-Server} = & H(\textit{Server Identifier}\,||\\ & \textit{Server HIMMO Credentials Length}\,||\\ & \textit{Server HIMMO-credentials}) \end{aligned}$$

  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

$$\begin{aligned} \textit{ID-Client} = H(&\textit{Client Identifier}\,||\\ &\textit{Client HIMMO Credentials Length}\,||\\ &\textit{Client HIMMO-credentials}) \end{aligned}$$

  Otherwise, set *ID-Client* = *Client HIMMO-Identifier*.
- Step 3: Compute the pairwise key $K_{\textit{ID-Client, ID-Server}}$ as shown in Equation 3.

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 Equation 1) 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 compute 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.

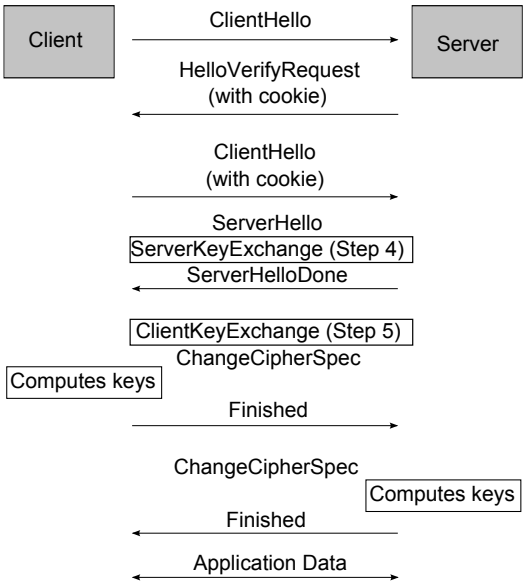

Figure 3. A DTLS-HIMMO exchange

6.3 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 to consider refers to the protection of the exchanged credentials that might contain some private information that is not wished to be exposed to the other party, if not authenticated before, or to any passive eavesdropper. The fact that HIMMO allows for identity-based key agreement allows for a simple extension of the DTLS-handshake. The credentials can be encrypted with the pairwise key shared with the other party. For instance, in the DTLS-HIMMO exchange, the client can protect 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 the encrypted client's HIMMO credentials. The server can use the HIMMO identifier to obtain the common pairwise key, and decrypt the client's credentials. Neither a fake server nor an attacker eavesdropping the communication will be 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 the one 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 the TTP) could eavesdrop or alter the ongoing data exchanges between any pair of nodes in a passive way. As explained in Section 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.

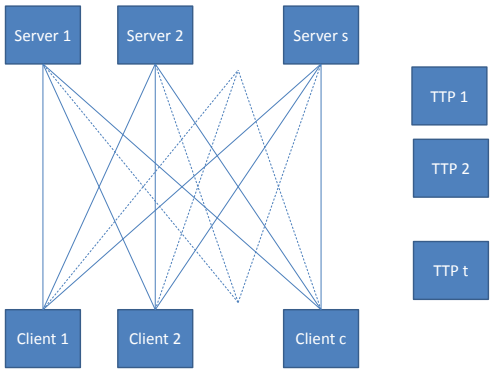

Figure 4. TTP infrastructure for privacy-protection in DTLS-HIMMO

6.4 TTP Infrastructure

The introduction of an infrastructure of TTPs (see Figure 4) 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 receive the corresponding key generating polynomials, each linked to the same or related credentials. Each entity would store these information either combined, as explained in Section 3.3, or independently. In this case, the TTP identifiers can be exchanged between client and server during a DTLS-HIMMO handshake as indicated in Figure 2. 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 common or chosen TTPs in the *ClientKeyExchange* messages.

Such an infrastructure brings new challenges but also advantages. Today, if a CA is compromised, then it is not possible to easily recover. Certificates are often not signed by more of a CA, and if they are, this rapidly increases the bandwidth and computational requirements. The usage of such a TTP Infrastructure as described above overcomes the problem of a TTP being compromised (and therefore the whole system being insecure) without almost any effect on bandwidth or computational resources.

7 DTLS-HIMMO AND COMPARISON WITH EXISTING (D)TLS ALTERNATIVES

We have implemented the DTLS-HIMMO operation mode in the CyaSSL library (version 3.0.0) 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 Section 6. 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 sever verification only and (vi) DTLS certificates with both server and client verification. Both modes are implemented using the ECDHE and ECDSA on NIST secp256r1 curve. 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 the reader with 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.

It is worth noting that the verification of the client or server credentials only costs an additional hash computation due to the identity-based nature of HIMMO. This is also the reason why the communication overhead can be kept at a very low level compared with certificates. At the same time, we also observe that the realization of the key agreement and verification

Table 4
Qualitative comparison of the HIMMO based PSK profile with other algorithms

DTLS mode	Client CPU Needs	Server CPU Needs	Handshake size	Certificate size	Key Agreement	Authentication	Information verification	Scalability
DTLS-HIMMO	1 HIMMO key generation (b=B)	1 HIMMO key generation (b=B) Key reconciliation	Low	Low	Yes	Mutual	No	$G_\xi(x)$ installation
DTLS-HIMMO(SA) (Server authentication)	1 SHA-256 1 HIMMO key generation (2b = B)	1 HIMMO key generation Key reconciliation (2b = B)	Low	Low	Yes	Mutual	Server authentication	$G_\xi(x)$ installation
DTLS-HIMMO (Mutual authentication)	1 SHA-256 1 HIMMO key generation (2b = B)	1 SHA-256 1 HIMMO key generation (2b = B) Key reconciliation	Low	Low	Yes	Mutual	Server and Client	$G_\xi(x)$ installation
PSK	-	-	Low	Low	Yes	Mutual	No	Installation of PSKs
ECDHE + ECDSA (Server authentication)	Three ECC point multiplications	One ECC point multiplication	High	High	Yes	Unilateral	Server verification	Root Certificate installation
ECDHE + ECDSA (Mutual authentication)	Three ECC point multiplications	Three ECC point multiplication	Higher	Higher	Yes	Mutual	Server and client verification	Root Certificate installation

of information requires several scalar ECC point multiplications while in the case of HIMMO only a polynomial evaluation is involved.

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 5 shows the required time to establish a secure connection and send the data for different cipher suites. We observe 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 $2b$ in this case. We also observe that the usage of a small or high security parameter α does not heavily impact the performance of the scheme remaining around a factor 8 faster than the ECC alternative. A value of α of 26 and 50 implies than an attacker has to deal with lattices of dimensions 405 and 1377 for the HI problem [5]. It is also worth noting that in all cases the cryptographic operations involved in the transfer of data are negligible compared with the DTLS handshake. Figure 6 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. Figure 7 shows the ratio between the required bandwidth and the exchanged payload making clear 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 Figure 1 show several advantages of HIMMO compared with other alternatives. The first one is that Internet of Things applications that involve the exchange of little data, frequently under 10 KB, can profit 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 one 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, Figures 5 and 7 show the performance of the DTLS handshake between two powerful devices. In a real world Internet of Things scenario one of those devices will have much lower capabilities. However, HIMMO can be still implemented in a very efficient way as illustrated in Figure 2.

There are other schemes that introduce post quantum secure key exchange. The post quantum key exchange scheme [1] is based on the ring learning with errors (R-LWE) problem and its authors discuss using it with TLS in a similar setup as the one discussed in this paper. Their results show that this scheme is slower than ECDH, for both client operation (from 0.8 msec to 1.4 msec) and for server operation (from 1.4 msec to 2.1 msec). Although these performance

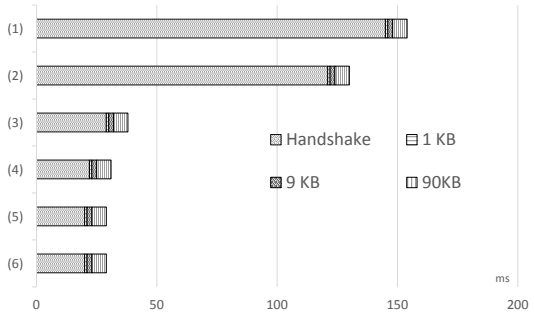

Figure 5. DTLS connection time. From top to bottom: (1) ECDH-ECDSA with mutual authentication, (2) ECDH-ECDSA with server authentication, (3) HIMMO with mutual verification of client's and server's credentials ($B = 256, b = 128, \alpha = 50$), (4) HIMMO with mutual verification of client's and server's credentials ($B = 256, b = 128, \alpha = 26$), (5) HIMMO with mutual authentication ($B = 128, b = 128, \alpha = 26$) and (6) PSK.

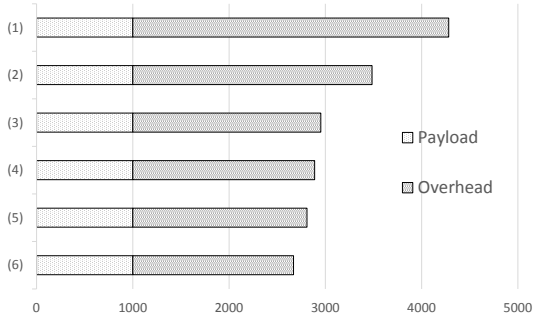

Figure 6. Total KB exchanged for 1 KB payload. From top to bottom: (1) ECDH-ECDSA with mutual authentication, (2) ECDH-ECDSA with server authentication, (3) HIMMO with mutual verification of client's and server's credentials, (4) HIMMO with verification of server's credentials, (5) HIMMO with mutual authentication and (6) PSK.

numbers are not directly comparable with ours since they are based on different CPUs and different software, it indicates that HIMMO should operate much faster than R-LWR since HIMMO is faster than ECC, and ECC is faster than the R-LWE scheme. PKC schemes based on NTRU have been investigated to secure Internet of Things scenarios [16]. NTRU is a fast scheme. However, the public-keys are long: a security level of 128-bits require 6743 bit long keys [14]. This also indicates that HIMMO and its identity-based nature can improveon NTRU communication overhead, very important in Internet of Things applications, while providing high speed performance.

8 Conclusions

The HIMMO scheme is the first Key Pre-distribution scheme that is simultaneously efficient and secure (in terms of collusion resistance). 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.

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 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

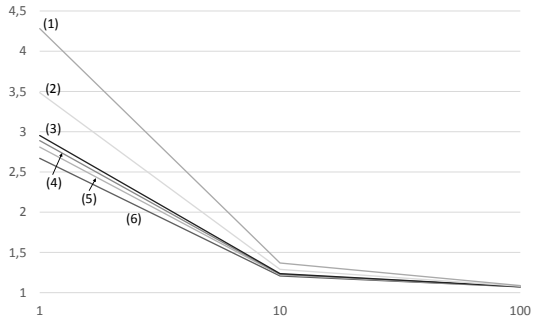

Figure 7. Ratio between total exchanged data and payload. From top to bottom: (1) ECDH-ECDSA with mutual authentication, (2) ECDH-ECDSA with server authentication, (3) HIMMO with mutual verification of client's and server's credentials, (4) HIMMO with verification of server's credentials,(5) HIMMO with mutual authentication and (6) PSK.

the standards, but offering a significantly improved performance security trade-off while being quantum secure. 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.

REFERENCES

[1] J.W. Bos, C. Costello, M. Naehrig, and D. Stebila. Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. Cryptology ePrint Archive, Report 2014/599, 2014. http://eprint.iacr.org/.

[2] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176.

[3] P. Eronen and H. Tschofenig. Pre-Shared Key Ciphersuites for Transport Layer Security (TLS). RFC 4279 (Proposed Standard), December 2005.

[4] O. García-Morchón, D. Gómez-Pérez, J. Gutiérrez, R. Rietman, and L. Tolhuizen. The MMO problem. In *Proc. ISSAC'14*, pages 186–193. ACM, 2014.

[5] O. García-Morchon, R. Rietman, I.E. Shparlinski, and L. Tolhuizen. Interpolation and approximation of polynomials in finite fields over a short interval from noisy values. *Experimental mathematics*, 23:241–260, 2014.

[6] O. García-Morchón, R. Rietman, L. Tolhuizen, D. Gómez-Pérez, and J. Gutiérrez. HIMMO - A Lightweight, Fully Colluison Resistant Key-Predistribution Scheme. Cryptology ePrint Archive, Report 2014/698, 2014. http://eprint.iacr.org/.

[7] O. Garcia-Morchon, L. Tolhuizen, D. Gomez, and J. Gutierrez. 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).*, pages 30–36, 2012.

[8] A. Liu and P. Ning. Tinyecc: A configurable library for elliptic curve cryptography in wireless sensor networks. In *Proc. 7th Int. Conf. on Information Processing in Sensor Networks*, IPSN '08, pages 245–256, Washington, DC, USA, 2008. IEEE Computer Society.

[9] T. Matsumoto and H. Imai. On the key predistribution system: a practical solution to the key distribution problem. In C. Pomerance, editor, *Advances in Cryptology – CRYPTO'87*, LNCS 293, pages 185–193. Springer, 1988.

[10] D. McGrew and D. Bailey. AES-CCM Cipher Suites for Transport Layer Security (TLS). RFC 6655 (Proposed Standard), July 2012.

[11] Ph.Q. Nguyen and B. Vallée, editors. *The LLL Algorithm - Survey and Applications*. Springer, 2010.

[12] J. Proos and C. Zalka. Shor's discrete logarithm quantum algorithm for elliptic curves. In *http://arxiv.org/abs/quantph/0301141*, 2003.

[13] Sage. http://www.sagemath.org.

[14] J. Schanck, 2014. https://github.com/NTRUOpenSourceProject/ntru-crypto/.

[15] Z. Shelby, K. Hartke, and C. Bormann. The Constrained Application Protocol (CoAP). RFC 7252 (Proposed Standard), June 2014.

[16] J.-R. Shih, Y. Hu, M.-C. Hsiao, M.-S. Chen, W.-C. Shen, B.-Y. Yang, A.-Y.Y Wu, and C.-M. Cheng. Securing m2m with post-quantum public-key cryptography. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 3(1):106–116, March 2013.

[17] P.W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

[18] H. Tschofenig. A Datagram Transport Layer Security (DTLS) 1.2 Profile for the Internet of Things, August 2014.

APPENDIX A
PROOF OF CORRECTNESS OF OPTIMIZED ALGORITHM

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

$$K := \langle \langle \sum_{i=0}^{\alpha} G_i \eta^i \rangle_N \rangle_{2^b}. \tag{4}$$

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

$$G_i = \gamma_i 2^{(\alpha-i-1)B+b} + \delta_i \text{ with } 0 \leq \delta_i \leq 2^{(\alpha-i-1)B+b} - 1. \tag{5}$$

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

In this appendix we prove the following theorem.

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

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

$$\Lambda_\alpha := G_\alpha$$

and for $0 \leq j \leq \alpha - 1$,

$$\Lambda_j := \eta \Lambda_{j+1} + G_j - \lfloor \frac{\sigma_j}{2^{(j+2)B}} \rfloor N.$$

By induction on j, it is easy to see that for $0 \leq j \leq \alpha$,

$$\Lambda_j \equiv \sum_{i=j}^{\alpha} G_i \eta^{i-j} \bmod N.$$

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,

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

As a consequence, if $\alpha < 2^B$, then $0 \leq \Lambda_0 - \tau_0 2^{\alpha B+b} < N$. The algorithm implies that $0 \leq \tau_0 \leq 2^B - 1$, and so $0 \leq \tau_0 \leq \Lambda_0 < N + 2^B - 1$. 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

$$K \in \{\langle \Lambda_0 \rangle_{2^b}, \langle \Lambda_0 + 1 \rangle_{2^b}\}. \tag{6}$$

In Proposition 3, we show that $\Lambda_j \equiv k_j$ for $0 \leq j \leq \alpha$. Combining this result with (6) proves the theorem.

For $0 \leq j \leq \alpha$, 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

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

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

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

and so

$$\lfloor \frac{\sigma_j}{2^{(j+2)B}} \rfloor \leq 2^B - 1. \tag{7}$$

According to (5), we have that $\delta_j \leq 2^{(\alpha-1-j)B+b} - 1$, and we clearly have that $\langle \sigma_j \rangle_{2^B} \leq 2^B - 1$. Combining these inequalities with (7) 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

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

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. $\square$