International Journal of Information Security (2018) 17:701–718
https://doi.org/10.1007/s10207-017-0397-6

REGULAR CONTRIBUTION

Comparing apples with apples: performance analysis of lattice-based authenticated key exchange protocols

Nina Bindel[1] · Johannes Buchmann[1] · Susanne Rieß[1]

Published online: 9 December 2017

Abstract
In view of the expected cryptanalysis (of both classical and quantum adversaries), it is important to find alternatives for currently used cryptographic primitives. In the past years, several authenticated key exchange protocols (AKE) that base their security on presumably quantum hard problems, such as lattice-based AKEs, were proposed. Since very different proposals for generic AKEs as well as direct AKEs, i.e., protocols directly based on lattice-based problems without additional authentication, exist, the performance of lattice-based AKEs is not evaluated and compared thoroughly. In particular, it is an open question whether the direct constructions are more efficient than generic approaches as it is often the case for other primitives. In this paper, we fill this gap. We compare existing lattice-based authenticated key exchange protocols, generic and direct. Therefore, we first find the most efficient suitable primitives to instantiate the generic protocols. Afterward, we choose parameters for each AKE yielding approximately 100 or 192 bits of security. We implement all protocols using the same libraries and compare the resulting performance. We find that our instantiation of the AKE by Peikert (PQCrypto, 2014) is the most efficient lattice-based AKE. Particularly, it is faster than the direct AKE by Zhang et al. (EUROCRYPT, 2015).

Keywords Lattice-based cryptography · Key exchange · Authenticated key exchange · Post-quantum cryptography

1 Introduction

In recent years, post-quantum cryptography, i.e., cryptographic primitives that are presumably secure against quantum algorithms, became a very active field of research. In particular, since the US-American National Security Agency (NSA) and the National Institute of Standards and Technology (NIST)[1] revealed their plans concerning post-quantum cryptography, the research on post-quantum cryptography got additional boost. Due to its good efficiency and high variety, lattice-based cryptography is one of the best candidates as a post-quantum alternative for currently used public key cryptography.

During the transition from classical to post-quantum cryptography, (authenticated) key exchange protocols are one of the first primitives that will be substituted for classical protocols. However, currently it is not clear which of the proposed lattice-based key exchange protocols is the most efficient one: Different generic constructions [24,37] exist, but to our knowledge those schemes were not instantiated, evaluated, and compared with each other until now. In particular, it is unclear how instantiations of generic constructions compare to direct lattice-based authenticated key exchange protocols such as [41].

In this paper, we close this gap. We instantiate the generic AKEs, implement the resulting constructions and the directly constructed AKEs in C++ using the same libraries, and choose parameters to achieve the same security level. Afterward, we compare their efficiency by comparing their running times and space, i.e., key sizes and bits of communication. We consider the following AKEs: the generic AKE by Fujioka et al. [24,25] (FSXY), the generic AKE by Peikert [37], and the direct one-pass and two-pass AKEs by Zhang et al. [41] (ZZDSD).[2]

To this end, we proceed as follows:

[1] In 2015, the NSA announced to start changing from classical to post-quantum cryptography [35]. In 2016, NIST started its preparations for its upcoming post-quantum standardization challenge [34].

✉ Nina Bindel
nbindel@cdc.informatik.tu-darmstadt.de

[1] Department of Computer Science, Technische Universität Darmstadt, Hochschulstraße 10, 64289 Darmstadt, Germany

[2] We do not consider the AKE proposed in [17] since the authors already instantiate their protocol with NTRU-based primitives and compare it to ZZDSD.

– We first choose the most efficient lattice-based primitives to instantiate the generic AKEs FSXY and the protocol by Peikert. This means essentially to choose the most efficient lattice-based key encapsulation mechanisms (KEM). Therefore, we construct KEMs corresponding to the following (unauthenticated) key exchange protocols (KEX): the protocol by Ding et al. [20] (DXL), by Bos et al. [10] (BCNS), and by Alkim et al. [4] (NewHope). The KEM by Peikert [37] is the only lattice-based KEM so far. Note that it is basically the same as the KEM constructed from [10]. Namely, BCNS is essentially an instantiation of Peikert's KEM. Furthermore, we also consider the public key encryption scheme (PKE) by Lyubashevsky et al. [32,33] (LPR) since it was suggested as an instantiation for the FSXY protocol by Fujioka et al. [24].
– We choose parameters for security levels of approximately 100 and 192 bit for the considered KEMs. Since we do not always reach the exact bit security of 100 and 192 bit, we also refer to them as low and high bit-security level. We implement each of the KEMs in C++ using the same libraries and measure the running times.
– Comparing the resulting running times and space, we find that the NewHope-based KEM is currently the most efficient one. Since the protocol NewHope was constructed to be more efficient than its predecessors, this is not overly surprising. However, our analysis explains whether the speedup results from the new design or implementation methods. We use the (IND-CPA secure) KEM based on NewHope to instantiate Peikert's protocol. The FSXY protocol uses one IND-CPA and one IND-CCA secure KEM. Therefore, using the transform by Fujisaki and Okamoto [26], we construct an IND-CCA secure KEM based on NewHope to instantiate FSXY.
– Finally, we choose parameters for the direct AKEs by Zhang et al. [41], implement the two AKEs in the same way as before, measure the running times, and compare those with the results of our instantiations of the generic AKEs.

We find that our instantiation of Peikert's AKE protocol is most efficient with respect to running times and space. Particularly, it is faster than the direct two-pass AKE by Zhang et al. [41]. We give detailed analyses of all measured running times in Sects. 6 and 8 and summarize our results in Table 9, Sect. 9. We also name the respective security assumptions and the security models the protocols are proved in, but we do not consider those any further in our comparison. All our software is in public domain.[3]

[3] Our software is available on https://www.cdc.informatik.tu-darmstadt.de/cdc/personen/nina-bindel.

1.1 Organization

After introducing notations in Sect. 2, we explain (authenticated) key exchange protocols in general and summarize the different building blocks and possible instantiations of the constructions in Sect. 3. We describe the constructed KEMs in detail in Sect. 4. In Sect. 5, we explain how we choose parameters for each of the considered KEMs. Our implementations and experimental results of the KEMs are given in Sect. 6. Afterward, we describe all considered AKEs in Sect. 7, i.e., we describe our instantiations of the generic AKEs and the one-pass and two-pass ZZDSD. In Sect. 8, we give the parameter sets of the ZZDSD protocols and our performance results of all AKEs. Finally, we summarize our analyses in Sect. 9.

2 Preliminaries

In this section, we describe notations, hardness assumptions, and cryptographic primitives used later on.

2.1 Notations

We define κ to be the security parameter. Let $n = 2^k \in \mathbb{N}$ for some $k \in \mathbb{N}$ and let $q \in \mathbb{N}$ be a prime such that $q = 1 \pmod{2n}$ if not stated otherwise. We denote the finite field $\mathbb{Z}/q\mathbb{Z}$ by $\mathbb{Z}_q$, and we identify an element in $\mathbb{Z}_q$ with its representative in $\left[-\lfloor \frac{q}{2} \rfloor, \lfloor \frac{q}{2} \rfloor\right] \cap \mathbb{Z}$ (denoted by $\pmod{q}$). We define $\mathcal{R} = \mathbb{Z}[x]/\langle x^n + 1\rangle$ and $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$. Let R be a ring, then $R^\times$ denotes the set of units in R.

Furthermore, we denote all polynomials by lower case letters (e.g., p), all column vectors by bold lower case letters (e.g., $\mathbf{v}$), and all matrices by bold upper case letters (e.g., $\mathbf{M}$). We write $\mathbf{M}^T$ as the transpose of the matrix $\mathbf{M}$. Let $p \in \mathcal{R}_q$ be a polynomial. Then, $\mathbf{p} \in \mathbb{Z}_q^n$ denotes its coefficient vector. When speaking of the length of vector $\mathbf{p} \in \mathbb{R}^n$, we mean the Euclidean norm $\|\mathbf{p}\|_2$, also denoted by $||\mathbf{p}||$. All logarithms in this paper are in base 2. For $x \in \mathbb{R}$, we denote by $\lfloor x \rceil$ the rounding operator such that $\lfloor x \rceil = \lfloor x + 0.5 \rfloor$. We refer to [38] for an introduction to (ideal) lattices.

Let M be a finite set. With $r \leftarrow_\$ M$, we denote that an element r is drawn uniformly at random from M. With $r \leftarrow \chi$, we denote that an element r is drawn according to the distribution χ. We denote the discrete Gaussian distribution with standard deviation σ by D_σ. Let $\mathbf{v} \in \mathbb{Z}^n$. We write $\mathbf{v} \leftarrow D_\sigma^n$ to denote that each coordinate of $\mathbf{v}$ is sampled from D_σ. Similarly, $a \leftarrow D_\sigma^n$ means that each coefficient of a polynomial a is sampled from D_σ.

2.2 Lattice-based hardness assumptions

All key exchange protocols considered in this paper base their security on the (ring) decisional learning with errors problem

(R-LWE) or the (ring) short integer solution problem (R-SIS). In the following, we define first the R-LWE distribution and afterward the R-LWE problem.

Definition 1 (R-LWE *Distribution*) Let n and $q \geq 2$ be non-negative integers, $s \in \mathcal{R}_q$, and χ be a distribution over $\mathcal{R}$. The R-LWE distribution is denoted by $\mathcal{D}_{s,\chi}$ and outputs pairs $(a, b = \langle a, s\rangle + e) \in \mathcal{R}_q \times \mathcal{R}_q$, where $a \leftarrow_{\$} \mathcal{R}_q$ and $e \leftarrow \chi$.

Definition 2 (R-LWE *Problem*) Let n and $q \geq 2$ be nonnegative integers, χ be a distribution over $\mathcal{R}$, and $s \leftarrow \chi$. Given m samples $(a_0, b_0), \ldots, (a_{m-1}, b_{m-1}) \in \mathcal{R}_q \times \mathcal{R}_q$, the decisional learning with errors problem over rings $\mathsf{R\text{-}LWE}_{n,m,q,\chi}$ is to decide whether (a_i, b_i) is chosen with the R-LWE distribution or uniformly random in $\mathcal{R}_q \times \mathcal{R}_q$ for $i = 0, \ldots, m-1$.

We write $\mathsf{R\text{-}LWE}_{n,m,q,\sigma}$ if the distribution χ is the discrete Gaussian distribution with standard deviation σ and mean zero. Brakerski et al. [11] prove that using pairs $(a, b = as + te \pmod q)$ with $t \in \mathbb{Z}^{\times}$, instead of $(a, b = as + e \pmod q)$, does not weaken the hardness of R-LWE.

One of the key exchange protocols considered in this paper uses a cryptographic primitive with security based on the R-SIS problem defined as follows.

Definition 3 ($\mathsf{R\text{-}SIS}_{n,m,q,\beta}$ *Problem*) Given $a_0, \ldots, a_{m-1} \leftarrow_{\$} \mathcal{R}_q$, find $s_0, \ldots, s_{m-1} \in \mathcal{R}$ such that $\sum_{i=0}^{m-1} a_i s_i = 0 \pmod q$ and $0 < ||\mathbf{s}|| \leq \beta$ for $\beta \in \mathcal{R}_{>0}$.

We refer to [31,38] for further details on the relations between different variants of R-SIS and R-LWE and respective reductions to lattice problems.

2.3 Cryptographic primitives

In the following, we define cryptographic primitives used in the key exchange protocols considered in this work. We do not define the primitives hash function, message authentication code (MAC), pseudorandom functions (PRF), and key derivation function (KDF), but refer to [24,29] for a detailed explanation.

A *public key encryption scheme* (PKE) $\mathcal{E}$ is defined via the following three probabilistic polynomial-time algorithms $\mathcal{E} = (\mathcal{E}.\mathsf{Gen}, \mathcal{E}.\mathsf{Enc}, \mathcal{E}.\mathsf{Dec})$ defined over the message space $\mathcal{M}$ and the cipher text space C:

$$\begin{aligned}
\mathcal{E}.\mathsf{Gen}(1^{\kappa}) &\rightarrow (\mathsf{sk}, \mathsf{pk}),\\
\mathcal{E}.\mathsf{Enc}(m, \mathsf{pk}) &\rightarrow c \in C,\\
\mathcal{E}.\mathsf{Dec}(c, \mathsf{sk}) &\rightarrow m \in \mathcal{M} \text{ or a decryption failure } \perp.
\end{aligned}$$

A *key encapsulation mechanism* (KEM) $\mathcal{K}$ is defined via the three probabilistic polynomial-time algorithms $\mathcal{K} = (\mathcal{K}.\mathsf{Gen}, \mathcal{K}.\mathsf{Encap}, \mathcal{K}.\mathsf{Decap})$ defined over the cipher text space C, the message space $\mathcal{M}$, and the key space K as follows:

$$\begin{aligned}
\mathcal{K}.\mathsf{Gen}(1^{\kappa}) &\rightarrow (\mathsf{dk}, \mathsf{ek}),\\
\mathcal{K}.\mathsf{Encap}(\mathsf{ek}) &\rightarrow (c, k) \in C \times K,\\
\mathcal{K}.\mathsf{Decap}(c, \mathsf{dk}) &\rightarrow k \in K \text{ or failure } \perp.
\end{aligned}$$

An IND-CPA secure KEM can be obtained from an IND-CPA secure PKE and vice versa by taking $K = \mathcal{M}$. Therefore, we write $(\mathsf{sk}, \mathsf{pk})$ instead of $(\mathsf{dk}, \mathsf{ek})$ and instantiate a KEM with a PKEin, e.g., Sect. 7. The cipher text c and the key k $(c, k) \leftarrow \mathcal{K}.\mathsf{Encap}(\mathsf{ek})$ correspond to a randomly chosen message m and the cipher text output $c \leftarrow \mathcal{E}.\mathsf{Enc}(m, \mathsf{pk})$ of the encryption algorithm. Moreover, $k \leftarrow \mathcal{K}.\mathsf{Decap}(c, \mathsf{dk})$ corresponds to $m \leftarrow \mathcal{E}.\mathsf{Dec}(c, \mathsf{sk})$, i.e., the key of the KEM is derived as a randomly chosen message of the PKE.

A *signature scheme* $\Sigma = (\Sigma.\mathsf{Gen}, \Sigma.\mathsf{Sign}, \Sigma.\mathsf{Ver})$ is defined via the following three algorithms over the message space $\mathcal{M}$:

$$\begin{aligned}
\Sigma.\mathsf{Gen}(1^{\kappa}) &\rightarrow (\mathsf{sk}, \mathsf{pk}),\\
\Sigma.\mathsf{Sign}(\mathsf{sk}, m) &\rightarrow \sigma,\\
\Sigma.\mathsf{Ver}(\mathsf{pk}, m, \sigma) &\rightarrow \{1, 0\};
\end{aligned}$$

accepting σ for m corresponds to 1, rejecting to 0.

3 Overview of authenticated key exchange protocols

A key exchange (KEX) is a cryptographic primitive to derive a shared secret key via a public network communication between a number of parties that do not share any secret information before [29]. We describe a key exchange protocol via the following algorithms: initiation, response, and finish. During the initiation, the sender computes an ephemeral public–secret key pair and sends the public key to the receiver. During the response, the receiver also computes a public–secret key pair and the shared secret key, and sends the public key (and eventually some additional information) to the sender. During the finish, the sender computes the shared secret key.

A KEX that also authenticates the identities of the involved parties is called authenticated key exchange protocol (AKE). More formally, in an AKE protocol each party has a static public–secret key pair where the static public key is certified with the party's identity. When running the protocol, each party generates an ephemeral secret key and, depending on that, a corresponding ephemeral public key. The public key is sent to the other parties. Each invocation of the protocol is called session and identified by a session identity denoted by sid. The session identity sid usually consists of public information such as the identities of the involved parties and

public keys. By I_S and I_R, we denote the identities of the sender and the receiver, respectively.

An AKE protocol is said to have perfect forward secrecy (PFS) if a compromise of its static keys does not lead to a compromise of previously established and deleted session keys. A protocol is said to have weak perfect forward secrecy (wPFS) if a compromise of its static keys does not lead to a compromise of previously established and deleted session keys of all sessions where the adversary did not interfere actively [30]. We describe an authenticated key exchange protocol via the following algorithms: (static) key generation, initiation, response, and finish (possibly for both, the sender and the receiver). During the (static) key generation, the long-term keys of sender and receiver are computed. During the initiation, the sender computes an ephemeral public–secret key pair and sends the public key to the receiver. During the response, the receiver also computes a public–secret key pair and sends the public key (and eventually some additional information) to the sender. During the finish of the sender, the sender computes the shared secret key. During the finish of the receiver, the receiver computes the shared secret key. Depending on the protocol, sometimes the receiver computes the shared secret key during the response, and hence, the finish of the receiver is not necessary anymore, e.g., [41].

In lattice-based authenticated key exchange protocols, *generic* constructions as well as *direct* constructions exist. *Generic* constructions such as [24,37] prove their security based on different cryptographic primitives such as signature schemes, encryptions scheme, key encapsulation mechanisms, or hash functions in general. Each of the primitives is instantiated by appropriate schemes. Figure 1 shows how the protocols depend on different building blocks. For example, the generic AKE by Peikert [37] consists of an IND-CPA secure KEM, a PRF, a signature scheme, a MAC, and a hash function. Additionally, we give possible instantiations of the building blocks. For example, the most efficient strongly unforgeable lattice-based signature scheme is BLISS [22]; hence, we choose BLISS as instantiation for Peikert's protocol. The construction by Fujioka et al. [24], abbreviated by FSXY, is based on an IND-CCA secure KEM, but due to the transform by Fujisaki and Okamoto [26] an IND-CPA secure KEM can be used.

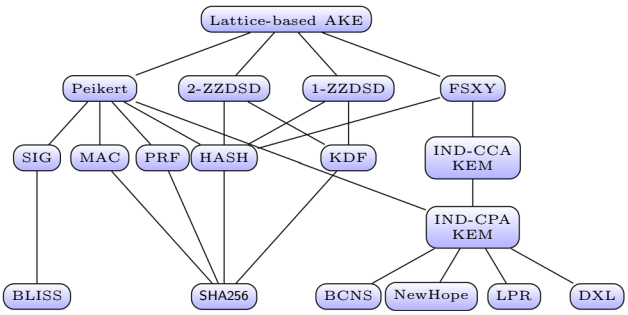

Fig. 1 AKEs and possible instantiations; abbreviations are as follows: Peikert is the AKE from [37], 2-ZZDSD and 1-ZZDSD are the AKEs from [41], FSXY is the AKE from [24], SIG stands for Signature scheme, MAC stands for message authentication code, PRF stands for pseudorandom function, HASH stands for hash function, KDF stands for key derivation function, KEM stands for key encapsulation mechanism, BLISS is the signature scheme from [22], SHA256 is a hash function instantiation, and BCNS is based on [10], NewHope on [4], LPR on [32,33], and DXL on [20]

As Fig. 1 indicates, there are different possibilities to instantiate the KEM in the protocols by Fujioka et al. and by Peikert. The only existing lattice-based KEM from the literature is proposed by Peikert [37]. However, Bos et al. [10] use the similarities of the construction of KEMs and KEXs to build the BCNS key exchange: The key exchange BCNS is essentially an instantiation of the KEM by Peikert. We use this relation the other way around and construct KEMs based on the most efficient key exchange protocols. Hence, we compare the performance of the KEMs based on the key exchange by Bos et al. [10] (BCNS), on NewHope by Alkim et al. [4], and on the protocol by Ding et al. [20] (DXL) in the next section. Fujioka et al. suggest to instantiate their protocol FSXY with the encryption scheme by Lyubashevsky et al. [32,33] (LPR). Hence, we consider a KEM based on LPR in the next section as well. Depending on the result of Sect. 6, we instantiate and compare the four AKEs.

In contrast to generic constructions, the security of *direct* constructions is directly based on some hard problem such as LWE. Also the direct constructions make use of different building blocks such as hash functions or key distribution functions (KDF), e.g., the two-pass and one-pass protocols of Zhang et al. [41] as we also summarize in Fig. 1. We abbreviate the protocols by 2-ZZDSD and 1-ZZDSD, respectively.

4 Lattice-based KEMs and signature schemes

In this section, we describe the KEMs built from (unauthenticated) key exchange protocols such as the DXL protocol [20], the BCNS protocol [10] which is essentially an instantiation of Peikert's KEM [37], and the NewHope protocol [4]. Since Fujioka et al. suggested it as a possible instantiation for the FSXY protocol, we describe the KEM based on LPR [32,33].

We depict the KEMs in detail in Figs. 2, 3 and 4. We use both terms, key exchange and KEM, for the protocols in this section when it is appropriate since they can be transformed into each other. Moreover, we use the same abbreviations, e.g., BCNS, for the key exchange protocol as well as for the KEM. We describe the protocols using S and R as abbreviations for the later used notions of *Sender* and *Receiver*, respectively.

$\mathcal{K}.\mathsf{Gen}(1^\kappa)$:
1. $s_S, e_S \leftarrow \chi$
2. $p_S = as_S + 2e_S$
3. return $((s_S, e_S), p_S)$

$\mathcal{K}.\mathsf{Decap}((c, p_R), (s_S, e_S))$
10. $e'_S \leftarrow \chi$
11. $v_S = s_S p_R + 2e'_S$
12. $SK_S = E(v_S, c)$
13. return SK_S

$\mathcal{K}.\mathsf{Encap}(p_S)$:
4. $s_R, e_R, e'_R \leftarrow \chi$
5. $p_R = as_R + 2e_R$
6. $v_R = p_S s_R + 2e'_R$
7. $c \leftarrow S(v_R)$
8. $SK_R = E(v_R, c)$
9. return $((c, p_R), SK_R)$

Fig. 2 KEM based on the DXL protocol [20]; computations are done in $\mathcal{R}_q$, functions E, and S as defined above, index S (resp., R) is used to indicate which of the values are computed by the sender (resp., the receiver) in the AKEs in Sect. 7

$\mathsf{Gen}(1^\kappa)$:
1. $s_S, e_S \leftarrow D^n_\sigma$
2. $p_S = as_S + e_S$
3. return $((s_S, e_S), p_S)$

$\mathsf{Decap}((c, p_R), s_S)$
11. $SK_S = rec(2p_R s_S, c)$
12. return SK_S

$\mathsf{Encap}(p_S)$:
4. $s_R, e_R, e'_R \leftarrow D^n_\sigma$
5. $p_R = as_R + e_R$
6. $v_R = p_S s_R + e'_R$
7. $\overline{v_R} \leftarrow dbl(v_R)$
8. $c = \langle \overline{v_R} \rangle_{2q,2}$
9. $SK_R = \lfloor \overline{v_R} \rceil_{2q,2}$
10. return $((c, p_R), SK_R)$

Fig. 3 KEM based on the BCNS protocol [10]; computations are done in $\mathcal{R}_q$, functions $\lfloor \cdot \rceil_{2q,2}$, $\langle \cdot \rangle_{2q,2}$, dbl, and rec as defined below; index S (resp., R) is used to indicate which of the values are computed by the sender (resp., the receiver) in the AKEs in Sect. 7

$\mathsf{Gen}(1^\kappa)$:
1. $seed \leftarrow_\$ \{0,1\}^{256}$
2. $a \leftarrow \mathsf{Parse}(H_1(seed))$
3. $s_S, e_S \leftarrow \Psi^n_k$
4. $p_S = as_S + e_S$
5. return $((s_S, e_S), (p_S, seed))$

$\mathsf{Decap}((c, p_R), s_S)$
14. $v_S = p_R s_S$
15. $k = Rec(v_S, c)$
16. $SK_S = H_2(k)$
17. return SK_S

$\mathsf{Encap}(p_S, seed)$:
6. $a \leftarrow \mathsf{Parse}(H_1(seed))$
7. $s_R, e_R, e'_R \leftarrow \Psi^n_k$
8. $p_R = as_R + e_R$
9. $v_R = p_S s_R + e'_R$
10. $c \leftarrow HelpRec(v_R)$
11. $k = Rec(v_R, c)$
12. $SK_R = H_2(k)$
13. return $((c, p_R), SK_R)$

Fig. 4 IND-CPA secure KEM based on the NewHope protocol [4]; computations are done in $\mathcal{R}_q$, functions Rec, and $HelpRec$ as defined above, for a fair comparison hash functions H_1 and H_2 are instantiated by SHA256 (Alkim et al. [4] use SHAKE-128 and SHA3-256), and index S (resp., R) is used to indicate which of the values are computed by the sender (resp., the receiver) in the AKEs in Sect. 7

4.1 Description of the KEM based on DXL

Ding et al. [20] propose a KEX that is secure against passive adversaries if R-LWE is hard. For a proof and an exact definition of this security model, we refer to [20]. The key exchange protocol is only proved to be secure in the two-user setting.

To describe the DXL protocol, we define the following functions below. Similar methods are also used in Sect. 7.3. Ding et al. [20] define the functions $\delta_0 : \mathbb{Z}_q \to \{0,1\}$, $\delta_1 : \mathbb{Z}_q \to \{0,1\}$, $S : \mathbb{Z}_q \to \{0,1\}$, and $E : \mathbb{Z}_q \times \{0,1\} \to \{0,1\}$ as

$$\delta_0(x) = \begin{cases} 0 & \text{if } x \in [-\lfloor \frac{q}{4} \rfloor, \lfloor \frac{q}{4} \rfloor] \\ 1 & \text{otherwise,} \end{cases}$$

$$\delta_1(x) = \begin{cases} 0 & \text{if } x \in [-\lfloor \frac{q}{4} \rfloor + 1, \lfloor \frac{q}{4} \rfloor + 1] \\ 1 & \text{otherwise,} \end{cases}$$

$$S(x) = \delta_b(x) \text{ with } b \leftarrow_\$ \{0,1\},$$

$$E(x, \delta) = x + \delta \frac{q-1}{2} (\text{mod } q, 2).$$

The functions δ_0, δ_1, S, and E can be extended to elements in $\mathcal{R}_q$.

We give the corresponding KEM in Fig. 2. The KEM depends on the parameters n, q, and an error distribution χ that fulfills

$$\Pr[\|x\| > \sqrt{2\pi n\sigma} : x \leftarrow \chi] \leq negl(n)$$

for some parameter σ. This holds for example for D^n_σ if $\sigma \geq \omega(\sqrt{\log(n)})$. Let $a \leftarrow_\$ \mathcal{R}_q$ be a publicly known value which can be shared among the parties.

The correctness of the scheme is guaranteed, i.e., it holds that $SK_S = SK_R$ with overwhelming probability, if $16\pi(n\sigma)^2 \leq \frac{q}{4} - 2$ [20]. Ding et al. propose the parameters $n = \lambda$, $q \approx \lambda^4$, and $\sigma = \frac{\lambda}{\sqrt{2\pi\lambda}}$.

4.2 Description of the KEM based on BCNS

Bos et al. [10] introduce a key exchange (BCNS) that replaces the traditional number theoretic key exchange in the Transport Layer Security protocol [18] by an R-LWE-based protocol.

The BCNS protocol is essentially an instantiation of the KEM introduced by Peikert [37], with m being a power of two and hence $g = 1$ in Peikert's KEM. We give the definitions of the needed functions for the instantiation used in the BCNS protocol. Let q be a positive modulus. Then, the modulus rounding function is defined as

$$\lfloor \cdot \rceil'_{q,2} : \mathbb{Z}_q \to \mathbb{Z}_2, x \mapsto \left\lfloor \frac{2}{q} x \right\rceil \pmod 2.$$

If q is an odd integer, the modulus rounding function on $\mathbb{Z}_q$ would be biased. Therefore, Peikert [37] introduced a randomized doubling function $dbl : \mathbb{Z}_q \to \mathbb{Z}_{2q}, x \mapsto 2x - e$, where $e = -1$ with probability 0.25, $e = 0$ with probability 0.5, and $e = 1$ with probability 0.25. Let $I_0 = \{0, 1, \ldots, \lfloor \frac{q}{4} \rceil - 1\}$ and $I_1 = \{-\lfloor \frac{q}{4} \rfloor, \ldots, -1\} = \{\lfloor \frac{3q}{4} \rceil, \ldots, q-1\}$ in $\mathbb{Z}_q$. The cross-rounding function is defined by

$$\langle v \rangle_{q,2} = \begin{cases} 0 & \text{, if } v \in I_0 \cup (I_0 + \frac{q}{2}) \\ 1 & \text{, if } v \in I_1 \cup (I_1 + \frac{q}{2}). \end{cases}$$

Let $I_1' = \{-\lfloor \frac{q}{2} \rfloor, \ldots, -1\}$ and $I_0' = \{0, 1, \ldots, \lfloor \frac{q}{2} \rceil - 1\}$, and let $E = \left[-\frac{q}{4}, \frac{q}{4}\right) \cap \mathbb{Z}$. If q is odd, the reconciliation function $rec : \mathbb{Z}_{2q} \times \mathbb{Z}_2 \to \mathbb{Z}_2$ with

$$rec(v, b) = \begin{cases} 0, & \text{if } v \in I_b' + E \pmod{2q}, \\ 1, & \text{otherwise} \end{cases}$$

is used to compute the shared key. The value b is also called the reconciliation information.

We give the corresponding KEM in Fig. 3. It depends on the parameters n, q, and σ where q is an odd prime and σ defines D_σ. Peikert uses a discretized Gaussian distribution, but we follow Bos et al. [10] and use a discrete Gaussian distribution. Let $a \leftarrow_\$ \mathcal{R}_q$ be a publicly known value which can be shared among the parties. The KEM based on BCNS, i.e., Peikert's KEM, is IND-CPA secure if R-LWE is hard given two R-LWE samples [37].

4.3 Description of the KEM based on NewHope

Alkim et al. [4] propose a generalization of the BCNS protocol, called NewHope. The main differences are a generalized reconciliation mechanism and a different error distribution. This allows new parameters with a much smaller modulus q. Alkim et al. [4] use the centered binomial distribution Ψ_k instead of the rounded Gaussian distribution without significantly decreasing the security of the protocol. This distribution has standard deviation $\sigma = \sqrt{k/2}$. We depict the KEM based on NewHope in Fig. 4.

Let $\mathbf{B} = (\mathbf{e_1}, \mathbf{e_2}, \mathbf{e_3}, \mathbf{g})$ where $\mathbf{g} = (1/2, 1/2, 1/2, 1/2)^T$ and $\mathbf{e_i}$ is the i-th vector of unity. Moreover, let D_4 be the lattice defined by $\mathbf{B}$.

The reconciliation and helper functions are defined as follows. If $\mathbf{v_S}^{(4)} \in \mathbb{Z}_q^4$ and $\mathbf{c}^{(4)} \in \{0, 1, 2, 3\}^4$, then

$$Rec\left(\mathbf{v_S}^{(4)}, \mathbf{c}^{(4)}\right) = \texttt{Decode}\left(\frac{1}{q}\mathbf{v_S}^{(4)} - \frac{1}{2^r}\mathbf{B}\mathbf{c}^{(4)}\right).$$

Moreover, the function $HelpRec(\mathbf{v_R})$ in the protocol of the receiver is defined as

$$HelpRec(\mathbf{v_R}^{(4)}) = \texttt{CVP}_{D_4}\left(\frac{2^r}{q}(\mathbf{v_R}^{(4)} + b\mathbf{g})\right) \pmod{2^r},$$

where $b \leftarrow_\$ \{0, 1\}$. The algorithms CVP and Decode are depicted in Figs. 5 and 6, respectively.

Furthermore, NewHope generates the public polynomial a pseudorandomly for every run of the KEM to prevent back doors and all-for-the-price-of-one attacks [4]. To generate a pseudorandom polynomial a, the authors use a 256 bit seed and a hash function, e.g., SHAKE-128 as suggested in [4]. This can be done similarly for all the other protocols. The generation of a is depicted in Fig. 4. However, it is assumed that a is publicly known in the other protocols. Hence, we do not consider the computation of a any further, especially not in Sect. 7.

$\texttt{CVP}_{D_4}(\mathbf{x})$:
1. $\mathbf{v_0} \leftarrow \lfloor \mathbf{x} \rceil$
2. $\mathbf{v_1} \leftarrow \lfloor \mathbf{x} - \mathbf{g} \rceil$
3. If $||\mathbf{x} - \mathbf{v_0}||_1 < 1$ set $k = 0$, else $k = 1$
4. $(v_0, v_1, v_2, v_3) \leftarrow \mathbf{v_k}$
5. return $\mathbf{z} = (v_0, v_1, v_2, k)^T + v_3(-1, -1, -1, 2)^T$

Fig. 5 CVP algorithm with input $\mathbf{x} \in \mathbb{R}^4$ and output $\mathbf{z} \in \mathbb{Z}^4$ such that $\mathbf{Bz}$ is a closest lattice point to $\mathbf{x}$

$\texttt{Decode}(\mathbf{x})$:
1. $\mathbf{v} = \mathbf{x} - \lfloor \mathbf{x} \rceil$
2. $k = 0$ if $||\mathbf{v}||_1 \leq 1$, else $k = 1$
3. return k

Fig. 6 Decode algorithm with input $\mathbf{x} \in \mathbb{R}^4/\mathbb{Z}^4$ and output $k \in \{0, 1\}$ such that $k\mathbf{g}$ is a closest vector to $\mathbf{x} + \mathbb{Z}^4$

CCA Secure NewHope-Based KEM. Following the explanations by Peikert [37], we describe an IND-CCA secure KEM based on the NewHope protocol. It is derived as follows: First, we construct an IND-CPA secure PKE from the IND-CPA secure NewHope-based KEM. Afterward, the Fujisaki-Okamoto transform (FOT) [26] is applied to derive an IND-CCA secure PKE. At last, an IND-CCA secure KEM based on NewHope is constructed from the IND-CCA secure PKE. We assume that the polynomial a is publicly known to all parties to simplify notations. The generation of a described above is still applicable. In addition to the function defined above, we define two more hash functions, $H_3 : \{0, 1\}^* \to \{0, 1\}^{256}$ and $H_4 : \{0, 1\}^* \to \{0, 1\}^{256}$, which can again be instantiated with SHA256.

A crucial point during the Fujisaki-Okamoto transform is to specify the randomness used during the encryption (resp., encapsulation). The encapsulation (i.e., the binomial sampler in Fig. 7) is run on randomness $PRG(H_3(r||k)))$ where $r, k \leftarrow_\$ \{0, 1\}^{256}$ and k is the session key. We depict the final IND-CCA secure KEM based on the NewHope protocol in Fig. 7.

4.4 Description of the KEM based on LPR

The LPR encryption scheme is based on the PKE described in [32,33]. The scheme depends on the parameters n, q, t, and k. Let $t = \omega(\sqrt{\lg n})$. Furthermore, let $\chi = \lfloor N(0, k^2/2\pi) \rceil$ be the rounded continuous Gaussian distribution with $k \leq \sqrt{\frac{q-2}{2(2n+1)2t^2}}$. This distribution is statistically indistinguishable from the discrete Gaussian distribution $D^n_{k/\sqrt{2\pi}}$. Let $a \leftarrow_\$ \mathcal{R}_q$ be a publicly known value which can be shared among the parties. We depict the corresponding KEM in Fig. 8.

$\mathsf{Gen}(1^\kappa)$:
1. $s_S, e_S \leftarrow \Psi_k^n$
2. $p_S = as_S + e_S$
3. return $((s_S, e_S), p_S)$

$\mathsf{Decap}((c = (c''||w, c'), p_R), s_S)$
16. $v_S = p_R s_S$
17. $k' = H_4(Rec(v_S, c'))$
18. $r \leftarrow c'' \oplus k'$
19. $k \leftarrow H_4(r) \oplus w$
20. $SK_S = H_2(k)$
21. return SK_S

$\mathsf{Encap}(p_S)$:
4. $k \leftarrow_\$ \{0,1\}^{256}$
5. $r \leftarrow_\$ \{0,1\}^{256}$
6. $s_R, e_R, e'_R \leftarrow \Psi_k^n$
7. $p_R = as_R + e_R$
8. $v_R = p_S s_R + e'_R$
9. $c' \leftarrow HelpRec(v_R)$
10. $k' = H_4(Rec(v_R, c'))$
11. $c'' \leftarrow k' \oplus r$
12. $w \leftarrow H_4(r) \oplus k$
13. $c = (c''||w, c')$
14. $SK_R = H_2(k)$
15. return $((c, p_R), SK_R)$

Fig. 7 IND-CCA secure KEM based on NewHope; computations are done in $\mathcal{R}_q$; index S (resp., R) is used to indicate which of the values are computed by the sender (resp., the receiver) in the AKEs in Sect. 7; functions Rec and $HelpRec$ are as defined above. In contrast to the CPA secure NewHope-based KEM in Fig. 4, we assume that the polynomial a is publicly known to all parties. The encapsulation is run with randomness $PRG(H_3(r||k)))$

$\mathsf{Gen}(1^\kappa)$:
1. $s_S, e_S \leftarrow \chi$
2. $p_S = as_S + e_S$
3. return $((s_S, e_S), p_S)$

$\mathsf{Decap}((c, p_R), s_S)$
10. $d = c - p_R s_S$
11. $k' = \lfloor \frac{2}{q} d \rceil \pmod 2$
12. return k'

$\mathsf{Encap}(p_S)$:
4. $s_R, e_R, e'_R \leftarrow \chi$
5. $p_R = as_R + e_R$
6. $v_R = p_S s_R + e'_R$
7. $k' \leftarrow_\$ \{0,1\}^n$
8. $c = v_R + \lfloor \frac{q}{2} k' \rceil$
9. return $((c, p_R), k')$

Fig. 8 KEM based on the LPR scheme [32,33]; computations are done in $\mathcal{R}_q$, index S (resp., R) is used to indicate which of the values are computed by the sender (resp., the receiver) in the AKEs in Sect. 7

4.5 Description of the signature scheme BLISS

The generic AKE protocol by Peikert [37] described in Sect. 7.2 uses a signature scheme which is strongly existentially unforgeable under chosen message attacks. We choose the signature scheme BLISS [22] since it is currently the most efficient lattice-based signature scheme satisfying the security condition of strong unforgeability. Its security is based on the hardness of R-SIS. We list other possible, but less efficient candidates in "Appendix A".

Since we do not modify the scheme BLISS, we do not describe it here but refer to [22].

5 Instantiation and choice of parameters

All protocols considered in this paper base their security on the hardness of R-LWE (except for the signature scheme BLISS which is based on R-SIS). Currently, the most efficient solvers for R-LWE for our instances are LWE solvers. Therefore, we use the LWE-Estimator by Albrecht, Player, and Scott [2] to estimate the hardness of our R-LWE instances.

Table 1 Overview of the parameters of all protocols for different bit-security levels

	DXL	BCNS	LPR	NewHope
Security (bit)	76	91	100	106
n	512	512	512	512
q	$\approx 2^{29}$	$\approx 2^{25.5}$	$\approx 2^{23.6}$	$\approx 2^{13.6}$
σ	3.19	3.19	3.19	3.46
Security (bit)	150	180	192	229
n	1024	1024	1024	1024
q	$\approx 2^{31}$	$\approx 2^{27}$	$\approx 2^{25.6}$	$\approx 2^{13.6}$
σ	3.19	3.19	3.19	2.82

In case this is not possible, we cite security estimations from other sources. All estimations by the tool are based on version f69b17a published on November 10, 2015.

The LWE-Estimator estimates the bit hardness of LWE against four kinds of attacks: the embedding approach, the decoding attack, the Blum–Kalai–Wassermann algorithm, and the Arora–Ge algorithm. We refer to [2] for more details.

The bit hardness of LWE depends on the dimension n, the modulus q, and the Gaussian distribution D_σ defined by $\alpha = \sqrt{2\pi}\frac{\sigma}{q}$. For $\alpha q \geq 8$, the continuous Gaussian distribution with $\sigma = \frac{\alpha q}{\sqrt{2\pi}} \geq \frac{8}{\sqrt{2\pi}}$ approximates the discrete Gaussian distribution well [41]. Therefore, we choose α (resp., σ) and q accordingly.

For each protocol, we aim to choose parameter sets with a bit security of 100 and 192 bit such that all correctness and security conditions given by the protocol definitions hold. As common [22,41], we assume that the security of the scheme is the same as the hardness of LWE, i.e., we implicitly assume that a tight security reduction from R-LWE to the scheme exists.

All protocols require n to be a power of two. Hence, the number of choices for n is limited and it is not always possible to reach exactly 100 or 192 bit. Increasing q while keeping the same values for α and n usually decreases the bit hardness of LWE. Moreover, higher values of q increase the running time of mathematical operations since the involved numbers become larger. Hence, we keep q as small as possible although we could have reached the exact bit hardness of 100 or 192 by increasing q. Since we do not always reach those bit-security levels exactly, we also refer to them as low and high bit-security level. Most of the correctness conditions give lower bounds for q. Since q depends linearly or quadratic on αq, we choose $\alpha q > 8$ as small as possible.

Since we aim for a fair performance comparison, we choose parameters under the same conditions as far as possible and state and explain our choices in the following. We summarize our instantiations and their respective bit security in Table 1.

5.1 Instantiation of DXL

By the correctness condition mentioned in Sect. 4.1, we obtain the following bound:

$$16\pi(n\sigma)^2 = 8(nq\alpha)^2 \leq q/4 - 2.$$

As described earlier, we choose $\alpha = 8/q$, yielding $q > 2048n^2 + 8$. With the smallest q that fulfills the above conditions, we obtain bit-security levels of 76 and 150 bit. For a bit security of 76 bit, we choose $n = 512$, $q = 536881153$, and $\sigma = 3.19$, called DXL-76. For bit security of 150 bit, we choose: $n = 1024$, $q = 2147493889$, and $\sigma = 3.19$, called DXL-150.

5.2 Instantiation of BCNS

The KEM based on BCNS is basically the KEM by Peikert [37] with n being a power of two. Hence, we choose parameters according to Peikert [37] to guarantee correctness, i.e., with $m = 2n$ the inequalities $q^2/64 \geq ((r^2 + 2\pi rad(m)/m)(2l^2 + n) + \pi/2)\omega^2$ and $\|s_S\|, \|e_S\| \leq l$ have to hold. This holds true for $l = n(r + \sqrt{rad(m)/m})\sqrt{n}$. For m being a power of two, $rad(m) = 2$. Hence, we get a lower bound on $q > \left(64\omega^2\left(\left(64 + \frac{2\pi}{n}\right)\left(128n^3 + 32n^{2.5} + 2n^2 + n\right) + \frac{\pi}{2}\right)\right)^{\frac{1}{2}}$, where $\omega > 0$ influences the probability of incorrectness, which is less than $2n\exp(3 \cdot 2^{-128} - \pi\omega^2)$. With $\omega = \sqrt{\frac{\ln(2n/\varepsilon)}{\pi}}$, we obtain an error probability of less than ε. The parameter q needs to be odd. We choose $\varepsilon = 2^{-128}$ and the smallest possible value for q and achieve security levels of 91 and 180. To obtain higher security levels (and hence to get closer to 100 or 192), a smaller q is required, and hence, the probability for incorrectness increases. For 91 bit of security, we choose $n = 512$, $q = 46565383$, $\sigma = 3.19$, and $\omega = 5.52$, called BCNS-91. For a bit security of 180, we choose $n = 1024$, $q = 131964963$, $\sigma = 3.19$, and $\omega = 5.54$, called BCNS-180.

5.3 Instantiation of NewHope

Alkim et al. [4] suggest two parameter sets. The first achieves a bit-security level of 106 bits. The other one achieves a bit-security level of 229 bits. We refer to these instantiations as NewHope-106 and NewHope-229, respectively. NewHope-106 is given by $q = 12289, n = 512$, and $\sigma = \sqrt{12}$ and NewHope-229 is given by $q = 12289, n = 1024$, and $\sigma = \sqrt{8}$. Since the secret is not chosen to be Gaussian distributed, we cannot use the LWE-Estimator to choose parameters closer to 192 bits of security.

5.4 Instantiations of LPR

According to the suggestions by Fujioka et al. [24], the LPR scheme requires $k \leq \sqrt{\frac{q-2}{2(2n+1)2t^2}}$ for $\chi \approx D^n_{k/\sqrt{2\pi}}$ and $k = \alpha q$. With $k \approx 8$, we obtain $256(2n+1)t^2 + 2 \leq q$. The value $t = \omega(\sqrt{\log_2(n)})$ influences the size of Gaussian sampled values. We choose t big enough so that we can remove the re-sampling step that Fujioka et al. [24] introduced.

We try different combinations of n and t, calculate the corresponding q, and test the bit security with the LWE-Estimator. The parameters that yield the desired bit security are $n = 512, t = 7, q = 12865537, \sigma = 3.19$ for 100 bit, called LPR-100, and $n = 1024, t = 10, q = 52457473, \sigma = 0.78$ for 192 bit, called LPR-192.

5.4.1 Instantiation of BLISS

Ducas et al. [22] state different parameter sets for their signature scheme BLISS. Since the LWE-Estimator does not give estimations for the hardness of R-SIS, we restate two parameter sets given by Ducas et al. [22] with the corresponding estimated security level. The parameter τ determines the repetition constant $M = \exp(\frac{1}{2\tau^2})$. For 128 bit of security, we use $n = 512$, $q = 12289$, $\sigma = 215$, $\delta_1 = 0.3$, $\delta_2 = 0$, $\tau = 1$, $\varsigma = 23$, $d = 10$, $B_2 = 12872$, and $B_\infty = 2100$, called BLISS-128. For 192 bit of security we use $n = 512$, $q = 12289$, $\sigma = 271$, $\delta_1 = 0.45$, $\delta_2 = 0.06$, $\tau = 0.55$, $\varsigma = 39$, $d = 8$, $B_2 = 9901$, and $B_\infty = 1613$, called BLISS-192.

6 Evaluating the performance

We implemented the KEMs mentioned in the previous section in C++ and compare the running times.[4] We use the hash functions from the cryptographic library from OpenSSL. Furthermore, we use the NTL library to implement the protocols. Our software is available on https://www.cdc.informatik.tu-darmstadt.de/cdc/personen/nina-bindel.

To measure the running time, we use `std::clock()`. Our results are given as the mean of 10,000 runs. Our experiments are done on a 3.60GHz Intel(R) Core(TM) i7-4790 CPU processor, 8GB RAM.

6.1 Running time of Gaussian sampling

We compare the timings of the following Gaussian sampling algorithms: rejection sampling [27], inverting the cumulative

[4] We do not consider the running times of the IND-CCA secure KEM based on NewHope in this section since the FOT is a generic transformation which can be applied to the other KEMs as well. Hence, it is enough to compare only the IND-CPA secure KEMs.

Table 2 Running times of Gaussian sampling algorithms

σ	Rejection	ICDF	Knuth–Yao	Ziggurat
3.19	4.78649	0.831372	0.262177	2.1033
425396	7.5628	2.60938	0.603896	2.00356

The times refer to the average running time of sampling $n = 2048$ elements from D_σ over 10,000 runs; running times are given in milliseconds (ms)

distribution function (ICDF) [36], the Ziggurat [12], and the Knuth–Yao algorithm [23]. We refer to the original papers for detailed descriptions.

We measure the average running time of sampling n elements from the discrete Gaussian distribution D_σ. In the analyzed protocols, the largest value of σ equals 425,396 and the smallest value of σ equals 3.19. Hence, we test all four Gaussian sampling algorithms for those two values of σ. The results are shown in Table 2. For both values of σ, the Knuth–Yao sampling method is most efficient, and hence, we use it in all protocols.

6.2 Running time of basic mathematical operations

In the following, we analyze the running time of small building blocks and mathematical operations of the different KEMs described in Sect. 3 with their respective instantiations for low and high bit-security levels.

The four mathematical operations which are used in each of the KEMs are sampling from the Gaussian distribution D_σ^n with the same standard deviation $\sigma = 3.19$ (or from a binomial distribution with an approximately the same standard deviation), multiplication of a polynomial and the integer 2, multiplication of two polynomials, and addition of two polynomials. Every mentioned mathematical operation occurs several times during one run of a protocol. Table 3 shows the average values over 10,000 runs of the complete protocol, and over every time the operation is computed during one run. For example, sampling from D_σ^n occurs four times during BCNS and hence, the stated time is the mean of those four measured running times which are already the means of 10,000 runs of the algorithm. Running times are rounded and given in milliseconds.

As indicated in Table 3, polynomial multiplication is the slowest mathematical operation in each scheme. It is ten to twelve times slower than Gaussian sampling, which is the second slowest operation. However, we emphasize that we do not use fast polynomial multiplication since the aim of this work is to give a fair comparison and not to provide efficient implementations. The fastest operation is the addition of two polynomials in all protocols except for NewHope, where sampling is the fastest operation.

Table 3 Running times of mathematical operations of the KEMs based on DXL, BCNS, LPR, and NewHope; running times are given in milliseconds (ms)

	DXL	BCNS	LPR	NewHope
Security (bit)	76	91	100	106
n	512	512	512	512
$\log_2(q)$	29	25.5	23.6	13.6
Gauss. sampling $\sigma = 3.19$	0.086	0.086	0.086	–
Bino. sampling $\sigma = 3.46$	–	–	–	0.007
2 times poly.	0.030	0.032	0.033	–
Poly. times poly.	0.870	0.875	0.876	0.869
Poly. addition	0.017	0.021	0.020	0.017
Security (bit)	150	180	192	229
n	1024	1024	1024	1024
$\log_2(q)$	31	27	25.6	13.6
Gauss. sampling $\sigma = 3.19$	0.158	0.159	0.159	-
Bino. sampling $\sigma = 2.82$	–	–	–	0.010
2 times poly.	0.058	0.062	0.064	–
Poly. times poly.	1.811	1.809	1.809	1.812
Poly. addition	0.034	0.042	0.039	0.034

We see that the running time of Gaussian sampling is approximately the same for all three protocols that use Gaussian sampling. The NewHope protocol replaces Gaussian sampling by sampling from a binomial distribution with the same standard deviation. For this operation, we cite the running times given in [4]. Hence, binomial sampling is roughly ten times faster than Gaussian sampling which might also be due to their much more efficient implementation.

The running time of multiplication of a polynomial with the integer 2 is rather different in the different KEMs because the multiplicated polynomials are distributed differently: In DXL the polynomials have Gaussian distributed coefficients. In the remaining protocols, the coefficients are uniformly distributed. A similar observation can be made for polynomial addition.

Each protocol additionally computes one or more other functions. This can be anything from dividing the modulus q by two, subtracting one from a constant involving q, or rounding a fraction. Those operations take between 0.0001 and 0.0002 ms and do not seem to depend significantly on the instantiation. Hence, we do not consider them any further.

6.3 Running time of reconciliation and helper functions

Aside from the four mathematical operations discussed in the previous section, each of the described KEMs, i.e., DXL, BCNS, NewHope, and LPR, contains some kind of reconciliation and helper function. We compare the running times

Table 4 Running times of corresponding reconciliation and helper functions of DXL, BCNS, NewHope, and LPR; running times are given in milliseconds (ms)

Protocol/instantiation	Helper function		Reconciliation	
	Function	Time	Function	Time
DXL-76	$S(\cdot)$	0.025	$E(\cdot,\cdot)$	0.036
BCNS-91	BCNS-91$\langle\cdot\rangle_{2q,2}$	BCNS-910.020	$rec(\cdot,\cdot)$	0.020
			$\lfloor\cdot\rceil_{2q,2}$	0.016
NewHope-106	$HelpRec(\cdot)$	0.142	$Rec(\cdot)$	0.010
LPR-100	Figure 8, L. 8	0.024	Figure 8, L. 10–11	0.070
DXL-150	$S(\cdot)$	0.050	$E(\cdot,\cdot)$	0.053
BCNS-180	$\langle\cdot\rangle_{2q,2}$	0.040	$rec(\cdot,\cdot)$	0.040
			$\lfloor\cdot\rceil_{2q,2}$	0.032
NewHope-229	$HelpRec(\cdot)$	0.283	$Rec(\cdot)$	0.020
LPR-192	Figure 8, L. 8	0.048	Figure 8, L. 10–11	0.141

Table 5 Running times of uniform sampling during DXL, BCNS, NewHope, and LPR for our instantiation of low and high bit-security levels; running times are given in milliseconds (ms)

Protocol	Sampling space	Running time (ms)	
		Low bit Sec.	High bit Sec.
DXL	$\{0,1\}^n$	0.022	0.044
BCNS	$\{-1,0,0,1\}^n$	0.026	0.051
NewHope	$\{0,1\}^{n/4}$	0.001	0.002
LPR	$\{0,1\}^n$	0.023	0.045

Table 6 Running times of the signature scheme BLISS; running times are given in milliseconds (ms)

Algorithm	BLISS-128	BLISS-192
Sign	0.124	0.375
Verify	0.030	0.032

of those in this section, listed in Table 4. The respective running times are rather small. The corresponding reconciliation function of DXL is E; the helper function is S. The corresponding reconciliation function for BCNS is $rec(\cdot,\cdot)$; the helper functions are $\langle\cdot\rangle_{2q,2}$ and $\lfloor\cdot\rceil_{2q,2}$. The corresponding reconciliation function for NewHope is $Rec(\cdot)$; the helper function is $HelpRec(\cdot)$. The corresponding reconciliation function for LPR is shown in Fig. 8, Lines 10 and 11; the helper function is shown in Fig. 8, Line 8.

6.4 Running time of the hash function and uniform sampling

All authenticated key exchange protocols need one or more hash functions. We always instantiate them with SHA256 and apply the hash function to elements of $\mathbb{Z}_q^x \times \{0,1\}^y$, where $x, y \in \mathbb{N}$. The running time depends on the value of x, y, and q. The running times increase with larger input, and hashing elements of $\mathbb{Z}_q$ takes longer than hashing bits. For example, hashing a 1024 bit string takes 0.01 ms while hashing an element in $\mathbb{Z}_q^{1024}$ takes between 0.035 and 0.05 ms depending on the size of q. Furthermore, Table 5 gives our results for sampling uniformly at random over different sampling spaces. Here the NewHope protocol performs very fast compared to the others because less intermediate data have to be saved. Similar methods might be possible for the other protocols.

6.5 Running time of the signature scheme BLISS

The authenticated key exchange protocol proposed by Peikert [37] also makes use of a signature scheme that we instantiate with BLISS [22]. Therefore, we recall the running time of BLISS in Table 6 according to Ducas et al. [22]. This means that the timings were not obtained on the same server via the same implementation. Since Ducas et al. give also a proof-of-concept implementation in the same language as we do, the measured times are comparable with our derived times. The running time of the key generation algorithm is not publicly available.

6.6 Overall running times

We summarize the overall running times and communication space in Table 7. The NewHope-based KEM is the fastest of the four analyzed KEMs. The main advantage lies in the key generation algorithm where it is about 0.15 and 0.3 ms (for the low and high bit-security level, respectively) faster than the others. This is caused by the faster sampling process. The second fastest KEM is the one based on LPR followed by BCNS. The differences, however, are small.

If we analyze the needed amount of communication bits, we find that NewHope needs to communicate approximately half of the other communication requirements. With this, we

Table 7 Overall running time and space of the four KEMs based on DXL, BCNS, NewHope, and LPR.

Protocol/instantiation	Security	Running time (ms)			Space
	(bit)	KeyGen	Encap	Decap	(bit)
DXL-76	76	1.089	2.176	1.039	31,232
BCNS-91	91	1.063	2.258	0.928	27,136
NewHope-106	106	0.09	1.948	0.890	15,360
LPR-100	100	1.065	2.101	0.995	36,864
DXL-150	150	2.219	4.427	2.114	66,560
BCNS-180	180	2.161	4.411	1.914	56,320
NewHope-229	229	1.866	4.028	1.837	30,720
LPR-192	192	2.163	4.280	2.047	79,872

conclude that NewHope is the most efficient instantiation of the four analyzed KEMs in terms of running time as well as communication bits.

Since Alkim et al. [4] state superiority of their scheme compared to all other schemes, our result does not come as a surprise. However, as Table 7 shows, the differences between NewHope and BCNS are not as large as stated in [4]: the encapsulation of BCNS-91 is only 1.16 times slower than NewHope, in contrast to eight times slower as shown in [4]. Hence, the advantage of NewHope during the encapsulation comes from very efficient implementation methods.

7 Description of AKE protocols

In this section, we describe four authenticated key exchange protocols: our instantiation of the generic AKE by Fujioka et al. [24], our instantiation of the generic AKE by Peikert [37], and the direct two-pass and one-pass AKE by Zhang et al. [41].

7.1 The FSXY protocol

Fujioka et al. [24] developed an AKE protocol, which is CK^+ secure [30] in the random oracle model.

Let $KEM_{cca} = (\mathsf{Gen}_{cca}, \mathsf{Encap}_{cca}, \mathsf{Decap}_{cca})$ and similarly $KEM_{cpa} = (\mathsf{Gen}_{cpa}, \mathsf{Encap}_{cpa}, \mathsf{Decap}_{cpa})$ be two key encapsulation mechanisms. Furthermore, let $H : \{0,1\}^* \to \{0,1\}^\kappa$ be a hash function. In the following, we describe the generic protocol, but we depict the protocol with our instantiation in Fig. 9.

Key generation: The sender (resp., receiver) generates $(\mathsf{sk}_S, \mathsf{pk}_S) \leftarrow \mathsf{Gen}_{cca}(1^\kappa)$ (resp., $(\mathsf{sk}_R, \mathsf{pk}_R)$).

Initiation: The sender generates the keys $(\mathsf{sk}_T, \mathsf{pk}_T) \leftarrow \mathsf{Gen}_{cpa}(1^\kappa)$, and it computes $(c_S, k_S) \leftarrow \mathsf{Encap}_{cca}(\mathsf{pk}_R)$. It sends (c_S, pk_T) to the receiver.

	Sender		Receiver
	Static keys		Static keys
	# $\mathsf{Gen}_{cca}(1^\kappa)$:		# $\mathsf{Gen}_{cca}(1^\kappa)$:
1	$s_S, e_S \leftarrow \Psi_k^n$		$s_R, e_R \leftarrow \Psi_k^n$
2	$p_S = as_S + e_S$		$p_R = as_R + e_R$
	# $\mathsf{Gen}_{cpa}(1^\kappa)$:		
3	$s_{S_1}, e_{S_1} \leftarrow \Psi_k^n$		
4	$p_{S_1} = as_{S_1} + e_{S_1}$		
	# $\mathsf{Encap}_{cca}(p_R)$):		
5	$k_S \leftarrow_\$ \{0,1\}^{256}$		
6	$r_S \leftarrow_\$ \{0,1\}^{256}$		
7	$s_{S_2}, e_{S_2}, e'_{S_2} \leftarrow \Psi_k^n$		
8	$p_{S_2} = as_{S_2} + e_{S_2}$		
9	$v_S = p_R s_{S_2} + e'_{S_2}$		
10	$c'_S \leftarrow HelpRec(v_S)$		
11	$k'_S = H_4(Rec(v_S, c'_S))$		
12	$c''_S \leftarrow k'_S \oplus r_S$		
13	$w_S \leftarrow H_4(r_S) \oplus k_S$		
14	$c_S = (c''_S \| w_S, c'_S)$		
15	$k_1 = H_2(k_S)$	$\xrightarrow[c_S, p_{S_2}]{p_{S_1}}$	# $\mathsf{Encap}_{cca}(p_S)$):
16			$k_{R_1} \leftarrow_\$ \{0,1\}^{256}$
17			$r_{R_1} \leftarrow_\$ \{0,1\}^{256}$
18			$s_{R_1}, e_{R_1}, e'_{R_1} \leftarrow \Psi_k^n$
19			$p_{R_1} = as_{R_1} + e_{R_1}$
20			$v_{R_1} = p_S s_{R_1} + e'_{R_1}$
21			$c'_{R_1} \leftarrow HelpRec(v_{R_1})$
22			$k'_{R_1} = H_4(Rec(v_{R_1}, c'_{R_1}))$
23			$c''_{R_1} \leftarrow k'_{R_1} \oplus r_{R_1}$
24			$w_{R_1} \leftarrow H_4(r_{R_1}) \oplus k_{R_1}$
25			$c_{R_1} = (c''_{R_1} \| w_{R_1}, c'_{R_1})$
26			$k_2 = H_2(k_{R_1})$
			# $\mathsf{Encap}_{cpa}(p_{S_1})$:
28			$s_{R_2}, e_{R_2}, e'_{R_2} \leftarrow \Psi_k^n$
29			$p_{R_2} = as_{R_2} + e_{R_2}$
30			$v_{R_2} = p_{S_1} s_{R_2} + e'_{R_2}$
31			$c_{R_2} \leftarrow HelpRec(v_{R_2})$
32			$k_{R_2} = Rec(v_{R_2}, c_{R_2})$
33		$\xleftarrow[c_{R_1}, p_{R_1}]{c_{R_2}, p_{R_2}}$	$k_3 = H_2(k_{R_2})$
	# $\mathsf{Decap}_{cca}(c_{R_1}, s_S)$:		#$\mathsf{Decap}_{cca}(c_S, s_R)$:
34	$v_{S_1} = p_{R_1} s_S$		$v_{R_3} = p_{S_2} s_R$
35	$k'_{R_1} = H_4(Rec(v_{S_1}, c'_{R_1}))$		$k'_S = H_4(Rec(v_{R_3}, c'_S))$
36	$r_{R_1} \leftarrow c''_{R_1} \oplus k'_{R_1}$		$r_S \leftarrow c''_S \oplus k'_S$
37	$k_2 \leftarrow H_2(H_4(r_{R_1}) \oplus w_{R_1})$		$k_1 \leftarrow H_2(H_4(r_S) \oplus w_S)$
	# $\mathsf{Decap}_{cpa}(c_{R_2}, s_{S_1})$:		
38	$v_{S_2} = p_{R_2} s_{S_1}$		
39	$k_{R_2} = Rec(v_{S_2}, c_{R_2})$		
40	$k_3 = H_2(k_{R_2})$		
	# Set session key:		# Set session key:
41	$SK = H(k_1, k_2, k_3, sid)$		$SK = H(k_1, k_2, k_3, sid)$

Fig. 9 FSXY authenticated key exchange protocol by Fujioka, Suzuki, Xagawa, and Yoneyama where the KEMs are instantiated by NewHope and hash functions by SHA256; we assume that the value a is publicly known to all parties; the session id is $sid = (I_S, I_R, p_S, p_R, p_{S_1}, c_S, c_{R_1}, c_{R_2})$

Response: The receiver computes values $(c_R, k_R) \leftarrow \mathsf{Encap}_{cca}(\mathsf{pk}_S)$ and $(c_T, k_T) \leftarrow \mathsf{Encap}_{cpa}(\mathsf{pk}_T)$, and it sends (c_R, c_T) to the sender. The receiver obtains $SK = H(k_S, k_R, k_T, sid)$ with $k_S \leftarrow \mathsf{Decap}_{cca}(c_S, \mathsf{sk}_R)$.

Finish: The sender computes $k_R \leftarrow \mathsf{Decap}_{cca}(c_R, \mathsf{sk}_S)$ and $k_T \leftarrow \mathsf{Decap}_{cpa}(c_T, \mathsf{sk}_T)$. It computes the session

key $SK = H(k_S, k_R, k_T, sid)$ with the session identity $sid = (I_S, I_R, \mathsf{pk}_S, \mathsf{pk}_R, \mathsf{pk}_T, c_S, c_R, c_T)$.

The FSXY protocol is correct, i.e., both parties compute the same shared session key, if both KEMs are correct. Furthermore, FSXY is CK^+ secure (in the random oracle model) if KEM_{cca} is one-way-CCA (OW-CCA) secure and KEM_{cpa} is one-way-CPA (OW-CPA) secure. We refrain from a formal security definition, since Fujioka et al. [24] instantiate KEM_{cpa} and KEM_{cca} by IND-CPA secure PKEs and transform those encryption schemes via the Fujisaki-Okamoto transform [26] (FOT) to IND-CCA secure encryption schemes. IND-CCA security implies OW-CCA security of the corresponding KEMs [26]. Let $\mathcal{E}.\mathsf{Enc}(m, \mathsf{pk})$ be the encryption function of an IND-CPA secure PKE $\mathcal{E}$ using randomness r. The FOT $\mathcal{E}'.\mathsf{Enc}(m, \mathsf{pk}) = \mathcal{E}.\mathsf{Enc}(m||r, \mathsf{pk})$ is the encryption function of an IND-CCA secure PKE $\mathcal{E}'$ using randomness $H(m||r)$, where H is a hash function. We refer to [26] for more details. Since the FOT is very simple, we assume that the most efficient KEM_{cpa} is transformed to the most efficient KEM_{cca}.

Due to our results in Sect. 6, we choose the IND-CPA and the IND-CCA secure KEM based on NewHope. We take a similar approach as Fujioka et al. to construct the CCA secure KEM based on NewHope using the FOT (cf. Fig. 7). We depict our instantiation of FSXY in Fig. 9.

7.2 The Peikert protocol

In the following, we describe the generic AKE by Peikert that we depict with our instantiation in Fig. 10. Peikert's protocol makes use of $KEM_{cpa} = (\mathsf{Gen}_{cpa}, \mathsf{Encap}_{cpa}, \mathsf{Decap}_{cpa})$ with key space K, $\Sigma = (\Sigma.\mathsf{Gen}, \Sigma.\mathsf{Sign}, \Sigma.\mathsf{Ver})$, $MAC = (\mathsf{MAC.Gen}, \mathsf{MAC}.\mathcal{O}, \mathsf{MAC.Ver})$ with key space K' and message space $\{0,1\}^*$, and a $PRF : K \times \{0,1\} \to K'$.

Key generation: The sender (resp., the receiver) generates $(\mathsf{sk}_S, \mathsf{pk}_S) \leftarrow \Sigma.\mathsf{Gen}(1^\kappa)$ (resp., $(\mathsf{sk}_R, \mathsf{pk}_R) \leftarrow \Sigma.\mathsf{Gen}(1^\kappa)$).

Initiation: The sender generates the ephemeral key pair $(s_S, p_S) \leftarrow \mathsf{Gen}_{cpa}(1^\kappa)$ and sends p_S to the receiver.

Response: The receiver computes the encapsulation $(c, k) \leftarrow \mathsf{Encap}_{cpa}(p_S)$, the values $k_0 = PRF(k, 0)$ and $k_1 = PRF(k, 1)$ with k_0 being the session key, and the MAC tag $t_R \leftarrow \mathsf{MAC}.\mathcal{O}(k_1, (1, sid, I_R))$ for message $(1, sid, I_R)$ and key k_1. Furthermore, the receiver generates the signature $\sigma_R = \Sigma.\mathsf{Sign}(\mathsf{sk}_R, (1, sid, p_S, c))$ with secret static key sk_R. The values t_R, σ_R, and c are sent to the sender.

Finish (sender): The sender computes the decapsulation $k \leftarrow \mathsf{Decap}_{cpa}(c, s_S)$ and the state $(k_0, k_1) = (PRF(k, 0), PRF(k, 1))$. The sender verifies the signature σ_R and the MAC tag t_R. If both are accepted, the sender accepts k_0 as session key. It computes the value $t_S \leftarrow \mathsf{MAC}.\mathcal{O}(k_1, (0, sid, I_S))$ and generates the signature $\sigma_S = \Sigma.\mathsf{Sign}(\mathsf{sk}_S, (0, sid, p_S, c))$. Both, t_S and σ_S, are sent to the receiver.

Finish (receiver): The sender verifies the signature σ_S and the MAC tag t_S. If both are accepted, k_0 is accepted as the session key.

Peikert's protocol is SK secure in the post-specified peer model [40] if the signature scheme Σ and MAC are existentially unforgeable under chosen message attacks, KEM_{cpa} is an IND-CPA secure KEM, and PRF is a secure pseudorandom function. SK security ensures, for example, weak perfect forward secrecy, and the post-specified peer model allows that the peering party is not specified in advance, but discovered during the run of the protocol. For more details, we refer to [14,37]. We instantiate the protocol with the lattice-based signature scheme BLISS and the IND-CPA secure KEM based on NewHope. The detailed protocol with our instantiation is given in Fig. 10. For reasons of clarity and comprehensibility, we do not give our instantiations of the hash functions or the MAC in Fig. 10, but we do instantiate both with SHA256 in our implementation.

7.3 The ZZDSD protocols

The AKE by Zhang, Zhang, Ding, Snook, and Dagdelen [41] is the only direct lattice-based AKE. There exists a two-pass and a one-pass version of the protocol which we call 2-ZZDSD and 1-ZZDSD, respectively. First, we define the needed cryptographic primitives and two additional functions. Afterward, the protocol is explained. The 2-ZZDSD and the 1-ZZDSD are depicted in detail in Figs. 11 and 12, respectively. Both protocols need a hash function and a KDF defined as

$$H : \{0,1\}^* \to D_{\sigma_3}^n \text{ and}$$
$$KDF : \{0,1\}^* \to \{0,1\}^n,$$

with σ_3 being one of three different standard deviations used in the protocol.[5] Furthermore, let $q > 2$ be an odd prime and define the set $E = \{-\lfloor \frac{q}{4} \rfloor, \ldots, \lfloor \frac{q}{4} \rceil\}$ and the functions

$$Cha : \mathbb{Z}_q \to \mathbb{Z}_2, v \mapsto \begin{cases} 0 & \text{if } v \in E, \\ 1 & \text{otherwise,} \end{cases}$$

and $Mod_2 : \mathbb{Z}_q \times \{0,1\} \to \{0,1\}$ with $Mod_2(v, b) = \left(\left(v + b\frac{q-1}{2}\right) \pmod q\right) \pmod 2$. The functions $Cha(\cdot)$ and $Mod_2(\cdot)$ can be extended to elements of $\mathcal{R}_q$ by applying them coefficient-wise to the image of elements of $\mathcal{R}_q$ under the coefficient embedding.

[5] In our implementation, we instantiate the hash function H by first using SHA256 and then using its random output bit string for sampling from $D_{\sigma_3}^n$

	Sender		Receiver
	Static signing and verification key		Static signing and verification key
	# Σ.Gen(1^κ)		# Σ.Gen(1^κ)
1	$f_S, g_S \leftarrow_\$ \mathcal{F}_{d_1,d_2}$		$f_R, g_R \leftarrow_\$ \mathcal{F}_{d_1,d_2}$
2	$(s_{S_1}, s_{S_2})^T \leftarrow (f_S, 2g_S + 1)^T$		$(s_{R_1}, s_{R_2})^T \leftarrow (f_R, 2g_R + 1)^T$
3	If $N_\kappa(\mathbf{S}_S) \geq 5C^2(d_1 + 4d_2)\kappa$:		If $N_\kappa(\mathbf{S}_R) \geq 5C^2(d_1 + 4d_2)\kappa$:
4	Restart		Restart
5	If f_S not invertible:		If f_R not invertible:
6	Restart		Restart
7	$a_S = (2g_S + 1)/f_S \pmod q$		$a_R = (2g_R + 1)/f_R \pmod q$
8	$pk_S = (2a_S, q - 2) \pmod{2q}$		$pk_R = (2a_R, q - 2) \pmod{2q}$
9	$sk_S \leftarrow (s_{S_1}, s_{S_2})^T$		$sk_R \leftarrow (s_{R_1}, s_{R_2})^T$
10	Return (s_S, p_S)		Return (s_R, p_R)
	# Gen$_{cpa}(1^\kappa)$		
11	$s_S, e_S \leftarrow \Psi_k^n$		
12	$p_S = as_S + e_S$	$\xrightarrow{p_S}$	# Encap$_{cpa}(p_S)$:
13			$s_R, e_R, e'_R \leftarrow \Psi_k^n$
14			$p_R = as_R + e_R$
15			$v_R = p_S s_R + e'_R$
16			$c \leftarrow HelpRec(v_R)$
17			$k' = Rec(v_R, c)$
18			$k = H_2(k')$
18			$(k_0, k_1) = (PRF(k, 0), PRF(k, 1))$
19			$t_R \leftarrow$ MAC.$\mathcal{O}(k_1, (1, sid, I_R))$
			# Σ.Sign(sk$_R$, $(1, sid, p_S, c)$):
20			$y_{R_1}, y_{R_2} \leftarrow D_\sigma^n$
21			$u_R = \zeta 2a_R y_{R_1} + y_{R_2} \pmod{2q}$
22			$c_R \leftarrow H\left(\lfloor u_R \rceil_d \pmod p, (1, sid, p_S, c)\right)$
23			$b_R \leftarrow_\$ \{0, 1\}$
24			$z_{R_1} \leftarrow y_{R_1} + (-1)^{b_R} s_{R_1} c_R$
25			$z_{R_2} \leftarrow y_{R_2} + (-1)^{b_R} s_{R_2} c_R$
26			Continue with probability $1/\delta_R$ otherwise restart
27			$z^\dagger_{R_2} \leftarrow (\lfloor u_R \rceil_d - \lfloor u_R - z_{R_2} \rceil_d \pmod p$
28			$\sigma_R = (z_{R_1}, z^\dagger_{R_2}, c_R)$
29	# Decap$_{cpa}(c, S_S)$:	$\xleftarrow{c, \sigma_R, t_R}$	
30	$v_S = p_R s_S$		
31	$k' = Rec(v_S, c)$		
32	$k = H_2(k')$		
33	$(k_0, k_1) = (PRF(k, 0), PRF(k, 1))$		
	# Σ.Ver(σ_R, p_R):		
34	If $\|(z_{R_1}\|2^d z^\dagger_{R_2})\|_2 > B_2$ or $\|(z_{R_1}\|2^d z^\dagger_{R_2})\|_\infty > B_\infty$:		
35	Reject		
36	If $c_R = H(\lfloor \zeta 2a_R z_{R_1} + \zeta q c_R \rceil_d + z^\dagger_{R_2} \pmod p, (1, sid, p_S, c))$:		
37	Accept		
38	MAC.Ver$(k_1, (1, sid, I_R), t_R)$		
39	$t_S \leftarrow$ MAC.$\mathcal{O}(k_1, (0, sid, I_S))$		
	# Σ.Sign(sk$_S$, $(0, sid, p_S, c)$):		
40	$y_{S_1}, y_{S_2} \leftarrow D_\sigma^n$		
41	$u_S = \zeta 2a_S y_{S_1} + y_{S_2} \pmod{2q}$		
42	$c_S \leftarrow H\left(\lfloor u_S \rceil_d \pmod p, (0, sid, p_S, c)\right)$		
43	$b_S \leftarrow_\$ \{0, 1\}$		
44	$z_{S_1} \leftarrow y_{S_1} + (-1)^{b_S} s_{S_1} c_S$		
45	$z_{S_2} \leftarrow y_{S_2} + (-1)^{b_S} s_{S_2} c_S$		
46	Continue with probability $1/\delta_S$ otherwise restart		
47	$z^\dagger_{S_2} \leftarrow (\lfloor u_S \rceil_d - \lfloor u_S - z_{S_2} \rceil_d \pmod p$		
48	$\sigma_S = (z_{S_1}, z^\dagger_{S_2}, c_S)$	$\xrightarrow{\sigma_S, t_S}$	# Σ.Ver(σ_S, p_S):
49			If $\|(z_{S_1}\|2^d z^\dagger_{S_2})\|_2 > B_2$ or $\|(z_{S_1}\|2^d z^\dagger_{S_2})\|_\infty > B_\infty$:
50			Reject
51			If $c_S = H(\lfloor \zeta 2a_S z_{S_1} + \zeta q c_S \rceil_d + z^\dagger_{S_2} \pmod p, (1, sid, p_S, c))$:
52			Accept
53			MAC.Ver$(k_1, (0, sid, I_S), t_S)$
	# Set session key:		# Set session key:
54	$SK_S = k_0$		$SK_R = k_0$

Fig. 10 Peikert's AKE where the KEM is instantiated with NewHope and the signature scheme with BLISS; $\mathcal{F}_{d_1,d_2} = \{h \in \mathcal{R}_q \mid d_1 = |\{h_i = \pm 1\}|, d_2 = |\{h_i = \pm 2\}|$, and the rest of the coefficients are equal to zero$\}$, $\delta_S = \left(M \exp(-||sk_S c_S||^2/(2\sigma^2)) cosh(\langle (z_{S_1}, z_{S_2})^t, sk_S c_S \rangle/\sigma^2)\right)$ and $\delta_R = \left(M \exp(-||sk_R c_R||^2/(2\sigma^2)) cosh(\langle (z_{R_1}, z_{R_2})^t, sk_R c_R \rangle/\sigma^2)\right)$; the function H_2 is the hash function from NewHope and H is the hash function from BLISS; computations are done in $\mathcal{R}_q$

	Sender		Receiver
	Static keys		Static keys
1.	$s_S, e_S \leftarrow D^n_{\sigma_1}$		$s_R, e_R \leftarrow D^n_{\sigma_1}$
	$p_S = as_S + 2e_S$		$p_R = as_R + 2e_R$
2.	$r_S, f_S \leftarrow D^n_{\sigma_2}$		$r_R, f_R \leftarrow D^n_{\sigma_2}$
	$x_S = ar_S + 2f_S$		$x_R = ar_R + 2f_R$
3.	$d_S = H(I_S, I_R, x_S)$		$sid = (I_S, I_R, x_S, x_R)$
	$\hat{r_S} = s_S d_S + r_S$		
	$\hat{f_S} = e_S d_S + f_S$		
4.	$z = (c(\hat{r_S}), c(\hat{f_S}))$		
	$z_1 = (c(s_S d_S), c(e_S d_S))$		
	Repeat steps 2-4 with probability $1 - \delta$		
5.		$\xrightarrow{x_S}$	$d_R = H(sid)$
			$\hat{r_R} = s_R d_R + r_R$
			$\hat{f_R} = e_R d_R + f_R$
6.			$z = (c(\hat{r_R}), c(\hat{f_R}))$
			$z_1 = (c(s_R d_R), c(e_R d_R))$
			Repeat steps 2, 6, 7 with probability $1 - \delta$
7.	$g_S \leftarrow D^n_{\sigma_2}$		$g_R \leftarrow D^n_{\sigma_2}$
	$d_R = H(sid)$		$d_S = H(I_S, I_R, x_S)$
	$h_S = (p_R d_R + x_R)\hat{r_S}$		$h_R = (p_S d_S + x_S)\hat{r_R}$
	$k_S = h_S + 2d_R g_S$		$k_R = h_R + 2d_S g_R$
8.		$\xleftarrow{x_R, w_R}$	$w_R = Cha(k_R)$
9.	$\sigma_S = Mod_2(k_S, w_R)$		$\sigma_R = Mod_2(k_R, w_R)$
	$sid = (sid, w_R, \sigma_S)$		$sid = (sid, w_R, \sigma_R)$
	$SK_S = KDF(sid)$		$SK_R = KDF(sid)$

Fig. 11 Protocol 2-ZZDSD with rejection probability $\delta = \min\left(\frac{D^{2n}_{\sigma_2}(z)}{MD^{2n}_{\sigma_2,z_1}(z)}, 1\right)$; computations are done in $\mathcal{R}_q$

	Sender		Receiver
	Static keys		Static keys
1.	$s_S, e_S \leftarrow D^n_{\sigma_1}$		$s_R, e_R \leftarrow D^n_{\sigma_1}$
	$p_S = as_S + 2e_S$		$p_R = as_R + 2e_R$
2.	$r_S, f_S \leftarrow D^n_{\sigma_2}$		
	$x_S = ar_S + 2f_S$		
3.	$d_S = H(I_S, I_R, x_S)$		
	$\hat{r_S} = s_S d_S + r_S$		
	$\hat{f_S} = e_S d_S + f_S$		
4.	$z = (c(\hat{r_S}), c(\hat{f_S}))$		
	$z_1 = (c(s_S d_S), c(e_S d_S))$		
	Repeat steps 2-4 with probability $1 - \delta$		
5.	$g_S \leftarrow D^n_{\sigma_2}$		
	$k_S = p_R \hat{r_S} + 2g_S$		
6.	$w_S = Cha(k_S)$		
7.		$\xrightarrow{x_S, w_S}$	$g_R \leftarrow D^n_{\sigma_1}$
			$d_S = H(I_S, I_R, x_S)$
			$k_R = (p_S d_S + x_S)s_R + 2d_S g_R$
8.	$\sigma_S = Mod_2(k_S, w_S)$		$\sigma_R = Mod_2(k_R, w_S)$
	$SK_S = KDF(sid)$		$SK_R = KDF(sid)$

Fig. 12 1-ZZDSD AKE protocol; the rejection probability is $1 - \delta$ with $\delta = \min\left(\frac{D^{2n}_{\sigma_2}(z)}{MD^{2n}_{\sigma_2,z_1}(z)}, 1\right)$; $sid = (I_S, I_R, x_S, w_S, \sigma_S) = (I_S, I_R, x_S, w_S, \sigma_R)$; computations are done in $\mathcal{R}_q$

Zhang et al. prove 2-ZZDSD to be secure in the Bellare–Rogaway security model [8]. If $16 \cdot 7\sigma_2^2\sqrt{n} < q$, correctness holds with overwhelming probability [41].

The protocol 1-ZZDSD is not wPFS. It is proved to be secure in a model similar to [30] in the random oracle model [41]. Also, for the 1-ZZDSD protocol a smaller q suffices for correctness. More concretely, q can be chosen $\frac{\sigma_2}{\sigma_1}$-times smaller than in the 2-ZZDSD protocol. More details follow in Sect. 8. The public parameters and the key generation algorithm are the same as in 2-ZZDSD.

Table 8 Running times of mathematical operations of 1-ZZDSD and 2-ZZDSD for the low and high bit-security level; running times are given in milliseconds (ms)

Protocol/ instantiation	2-ZZDSD -100	1-ZZDSD -81	2-ZZDSD -210	1-ZZDSD -160
Security (bit)	100	81	210	160
Gauss. sampl. σ_1	0.157	0.083	0.305	0.157
Gauss. sampl. σ_2	0.279	0.139	0.726	0.283
2 times poly.	0.060	0.030	0.117	0.059
Poly. mult.	2.380	0.869	5.147	1.814
Poly. addition	0.033	0.017	0.066	0.034
Cha	0.031	0.020	0.061	0.039
Mod_2	0.033	0.038	0.066	0.077

8 Instantiations and performance results of the AKEs

In this section, we give concrete instantiations of the ZZZDSD protocols. Afterward, we compare the performance of our C++ implementation of those protocols to our instantiation of FSXY and Peikert's protocol.

8.1 Concrete instantiation of ZZDSD

As before, we use the LWE-Estimator (or if not possible cite from the original source) to choose parameters for low (aiming to reach 100) and high (aiming to reach 192) bit-security levels for 1-ZZDSD and 2-ZZDSD. We summarize the running times of the main mathematical operations in Table 8.

The two-pass ZZDSD protocol requires n to be a power of two and an odd prime q such that $q = 1 \pmod{2n}$. For correctness of the protocol, $q > 16 \cdot 7\sigma_2^2\sqrt{n}$ with $\sigma_2 = \frac{1}{2}\tau\sigma_1^2 n$ has to hold. The parameter $\sigma_1 = \alpha q/\sqrt{2\pi}$ denotes the Gaussian parameter. The repetition rate for the rejection sampling is given by $M = \exp(\frac{12}{\tau} + \frac{1}{2\tau^2})$. With $\tau = 12$, we obtain $M = 2.72$, $n = 1024$, $q = 14186338877441$, $\sigma_1 = 3.192$, $\sigma_2 = 62914.56 \approx 2^{15.9}$, and $\sigma_3 = 3.397$ for which the LWE-Estimator returns a bit security of 100 bit, referred to as 2-ZZDSD-100. For the high bit-security level, we choose $n = 2048$. Since the tool fails to return a result, we restate the estimations given by Zhang et al. [41] for 210 bit of security. That means $q = 1125899906949121$, $\sigma_1 = 3.397$, $\sigma_2 = 425396.146$, $\sigma_3 = 3.397$, $\tau = 36$, and $M = 1.396$, called 2-ZZDSD-210.

In the one-pass ZZDSD protocol, q can be chosen such that $q > 112\sigma_2\sigma_1\sqrt{n}$. For $n = 512$, we obtain a bit security of 81 with $\tau = 12$, $q = 255111169$, $\sigma_1 = 3.2$, $\sigma_2 = 31457.28$, $\sigma_3 = 3.192$, and $M = 2.728$. Smaller values of τ increase the bit security, but also the rejection constant M to a value larger than 3. This is not practical. For $n = 1024$, we obtain a bit-security level between 151 and 160 bit for values of τ between 12 and 36. For $n = 2048$ and $\tau = 36$, the bit-security level is 313 bit. Hence, bit-security levels of 81 and 160 bit are the best we achieve. We choose the following parameters: $n = 1024$, $q = 721563649$, $\sigma_1 = 3.2$, $\sigma_2 = 62914.56$, $\sigma_3 = 3.192$, $\tau = 12$, and $M = 2.728$ for a security level of 160 bit, called 1-ZZDSD-160, and $n = 512$, $q = 255111169$, $\sigma_1 = 3.2$, $\sigma_2 = 31457.28$, $\sigma_3 = 3.192$, $\tau = 12$, and $M = 2.728$ for a bit-security level of 81 bit, called 1-ZZDSD-81.

8.2 Performance results of the AKEs

In this section, we present the results of our experiments of the ZZDSD protocols and the instantiations of the generic protocols. We measure the running times of the single operations during the run of the ZZDSD protocols in Sect. 8.1. As for the generic protocols FSXY and Peikert's protocol, which are constructed from building blocks as stated in Fig. 1, we use the results derived in Sects. 6 and 7. We obtain the total running times by adding the running times of their building blocks and state our results of the total running times in Table 9.

The overall running time is divided into three algorithms: key generation and the protocol of the receiver and the sender. We do not measure and consider the time to generate the polynomial a, but we assume that it is publicly known to all parties.

As indicated in Table 9, 1-ZZDSD outperforms the other three protocols clearly in its amount of communication bits for the low and high bit-security level. This is not too much of a surprise since it is a one-pass protocol. For protocols with two or more passes, our instantiation of Peikert's protocol uses the least communication bits for the low (resp., high) bit-security level with 27,341 (resp., 44,544) bits, followed first by the FSXY, and then by the ZZDSD protocol.

We compute the communicated bits as the total amount of communication bits, i.e., public keys and other transmitted data, such as reconciliation bits.

We estimate the public key size by $n\lceil\log_2(q)\rceil$ for $pk \leftarrow_\$ \mathcal{R}_q$ and take $2n\sigma$ as an estimation for the secret key size for a Gaussian (or binomial) sampled polynomial of degree n with standard deviation σ. The lowest public key size is given by Peikert with 7000 bits for approximately 106 bit and 192 bit of security. Similarly, the lowest secret key size is given by Peikert with 2000 bits for 106 bit security and 3000 bits for 192 bit of security.

Table 9 Overall running times and communication bits

Protocol/instantiation	Security	Total running time (ms)			Space (bit)			Message-	Hardness	Security model	(w)PFS?
	(bit)	KeyGen	Receiver	Sender	Communication	Public key	Secret key	passes	assumption		
FSXY-106	106	0.09	4.999	4.031	32,768	7168	3547	2-pass	R-LWE	CK^+ in ROM	wPFS
2-ZZDSD-100	100	2.782	29.296	29.268	91,136	45,056	6537			BR in ROM	
1-ZZDSD-81	81	1.084	2.830	9.280	14,848	14,336	3277	1-pass		auth. IND-CCA	–
Peikert-106	106	–	2.133	1.165	27,341	7168	2048	3-pass	R-LWE, R-SIS	SK	PFS
FSXY-229	229	1.866	10.167	9.843	64,512	14,336	5792	2-pass	R-LWE	CK^+ in ROM	wPFS
2-ZZDSD-210	210	5.904	41.104	41.047	210,944	104,448	13,914			BR in ROM	
1-ZZDSD-160	160	2.221	5.882	19.289	31,744	30,720	6554	1-pass		auth. IND-CCA	–
Peikert-192	192	–	4.467	4.143	44,544	7168	3072	3-pass	R-LWE, R-SIS	SK	PFS

Times are given in milliseconds (ms). Communication bits are given as the total amount of communication in byte. We write "–" as running time of the key generation of Peikert-106 and Peikert-192 since the running times depend on the time for key generation of the signature scheme BLISS for which those times are not publicly available. The abbreviation (w)PFS stands for (weak) perfect forward secrecy

Table 10 Overview of selected state-of-the-art lattice-based signature schemes that are strongly unforgeable under chosen message attack; sizes are given in byte; content of the table is taken from [3]

Scheme	Security (bit)	CPU	Key Size (byte)	Sign. Size (byte)	Cycle counts
GPV	96	AMD Opteron	vk: 28,508,160	30,105	sign: 312,800,000
[6,27]		8356 (Barcelona)	sk: 12,353,536		verify: 50,600,000
BG	97	Intel Core	vk: 1,619,940	1495	sign: 1,203,924
[5,16]		i7-4770K (Haswell)	sk: 912,380		verify: 335,072
GPV-poly	96	AMD Opteron	vk: 55,906	32,972	sign: 80,500,000
[6,27]		8356 (Barcelona)	sk: 26,316		verify: 11,500,000
BLISS- BI	128	"Intel Core	vk: 7168	1559	sign: ≈ 358,400
[21,22]		3.4GHz"	sk: 2048		verify: 102,000

The fastest total running time is given by our instantiation of Peikert's protocol. The running time of FSXY is approximately two to three times slower than Peikert's protocol. The 2-ZZDSD protocol is much slower than all other protocols. Its running time is, even without counting the time for repetition because of rejection sampling, three times slower than FSXY. As shown in [41], the running time of both ZZDSD protocols can be significantly reduced by choosing parameters for a lower repetition rate. However, this also reduces the bit security and does still not achieve a performance similar to our instantiation of Peikert's protocol.

9 Summary and evaluation

In this paper, we analyzed four different KEMs based on the following PKEs or KEX protocols: DXL, BCNS, NewHope, and LPR. We chose suitable parameters for each of the protocols aiming for 100 and 192 bits of security. We implemented each of the KEMs and compared the results of our experiments. Depending on this analysis we chose the KEM based on NewHope to instantiate the generic AKEs by Peikert and the FSXY protocol. We compared the resulting running time and space with our implementation of the 2-ZZDSD and 1-ZZDSD protocol which are directly constructed from lattice-based hardness assumptions.

We show that our instantiation of Peikert's AKE is more efficient with respect to running time and space compared to the direct construction of 2-ZZDSD. We give a detailed comparison of the four AKEs with respect to security and performance properties in Table 9.

We emphasize that we do not consider the differences between the security properties of the considered protocols in our performance analysis since this work is purely focused on a comparison with respect to the implementation methods. To compensate, we state the different security properties such as the security model and whether the protocol is forward secure in Table 9. One of the strongest security models for authenticated protocols is the CK^+ security model, which is used in the FSXY protocol. Unlike the other AKEs, 1-ZZDSD is not wPFS. This is because it has only one message pass. We leave the performance analysis with respect to security models for future work.

Acknowledgements We thank the anonymous reviewers for their detailed and helpful comments on an earlier version of this paper. This work has been co-funded by the DFG as part of project P1 within the CRC 1119 CROSSING.

A Comparison of selected signature schemes from the literature

In this section, we give a short overview on post-quantum signature schemes that are (not) suitable as instantiation for the AKE by Peikert. A suitable signature scheme must be strongly unforgeable under chosen message attack. We compare the performance of existing lattice-based signature schemes that are strongly unforgeable in Table 10. Other lattice-based signature schemes such as [1,3,7,28] are not proved to be strongly unforgeable, but only existentially unforgeable. The hash-based signature schemes SPHINCS [9] and XMSS [13] are also not proved to be *strongly* unforgeable. Multivariate signature schemes such as [15,19,39] are also not proved to be strongly secure, but only to be *globally unforgeable*. Hence, we choose BLISS to instantiate the AKE by Peikert, since BLISS is the most efficient scheme with respect to running times and sizes that fulfills the security requirements.

References

1. Akleylek, S., Bindel, N., Buchmann, J., Krämer, J., Azzurra Marson, G.: An efficient lattice-based signature scheme with provably secure instantiation. In: Progress in Cryptology—AFRICACRYPT 2016—8th International Conference on Cryptology in Africa, Fes, Morocco, 3–15 April 2016, Proceedings, pp. 44–60 (2016)
2. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. J. Math. Cryptol. **9**(3), 169–203 (2015)
3. Alkim, E., Bindel, N., Buchmann, J., Özgür Dagdelen, Eaton, E., Gutoski, G., Krämer, J., Pawlega, F.: Revisiting TESLA in the quantum random oracle model. Cryptology ePrint Archive, Report 2015/755 (2015). http://eprint.iacr.org/2015/755
4. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange—a new hope. In: 25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, 10–12 August 2016, pp. 327–343 (2016)
5. Bai, S., Galbraith, S.D.: An improved compression technique for signatures based on learning with errors. In: Benaloh, J. (ed.) CT-RSA 2014, LNCS, vol. 8366, pp. 28–47. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-319-04852-9_2
6. Bansarkhani, R.E., Buchmann, J.: Improvement and efficient implementation of a lattice-based signature scheme. In: Lange et al. [30], pp. 48–67. https://doi.org/10.1007/978-3-662-43414-7_3
7. Barreto, P., Longa, P., Naehrig, M., Ricardini, J., Zanon, G.: Sharper ring-lwe signatures. Cryptology ePrint Archive, Report 2016/1026 (2016). http://eprint.iacr.org/2016/1026
8. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) CRYPTO'93, LNCS, vol. 773, pp. 232–249. Springer, Heidelberg (1994)
9. Bernstein, D.J., Hopwood, D., Hülsing, A., Lange, T., Niederhagen, R., Papachristodoulou, L., Schneider, M., Schwabe, P., Wilcox-O'Hearn, Z.: SPHINCS: practical stateless hash-based signatures. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I, LNCS, vol. 9056, pp. 368–397. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_15
10. Bos, J.W., Costello, C., Naehrig, M., Stebila, D.: Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. In: 2015 IEEE Symposium on Security and Privacy, pp. 553–570. IEEE Computer Society Press, San Jose, CA, USA (2015). https://doi.org/10.1109/SP.2015.40
11. Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-LWE and security for key dependent messages. In: Rogaway, P. (ed.) CRYPTO 2011, LNCS, vol. 6841, pp. 505–524. Springer, Heidelberg (2011)
12. Buchmann, J., Cabarcas, D., Göpfert, F., Hülsing, A., Weiden, P.: Discrete ziggurat: a time-memory trade-off for sampling from a Gaussian distribution over the integers. In: Lange et al. [30], pp. 402–417. https://doi.org/10.1007/978-3-662-43414-7_20
13. Buchmann, J.A., Dahmen, E., Hülsing, A.: XMSS: a practical forward secure signature scheme based on minimal security assumptions. In: Yang, B. (ed.) Post-Quantum Cryptography—4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29–December 2 2011. Proceedings, Lecture Notes in Computer Science, vol. 7071, pp. 117–129. Springer (2011)
14. Canetti, R., Krawczyk, H.: Security analysis of ike's signature-based key-exchange protocol. In: Advances in Cryptology—CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, 18-22 August 2002, Proceedings, pp. 143–161 (2002)
15. Chen, A.I.T., Chen, M.S., Chen, T.R., Cheng, C.M., Ding, J., Kuo, E.L.H., Lee, F.Y.S., Yang, B.Y.: SSE implementation of multivariate PKCs on modern x86 CPUs. In: Clavier, C., Gaj, K. (eds.) CHES 2009, LNCS, vol. 5747, pp. 33–48. Springer, Heidelberg (2009)
16. Dagdelen, Ö., Bansarkhani, R.E., Göpfert, F., Güneysu, T., Oder, T., Pöppelmann, T., Sánchez, A.H., Schwabe, P.: High-speed signatures from standard lattices. In: Aranha, D.F., Menezes, A. (eds.) LATINCRYPT 2014, LNCS, vol. 8895, pp. 84–103. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-319-16295-9_5
17. del Pino, R., Lyubashevsky, V., Pointcheval, D.: The Whole is Less Than the Sum of Its Parts: Constructing More Efficient Lattice-Based AKEs, pp. 273–291. Springer International Publishing, Cham (2016)
18. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard) (2008). http://www.ietf.org/rfc/rfc5246.txt. Updated by RFCs 5746, 5878, 6176
19. Ding, J., Schmidt, D.: Rainbow, a new multivariable polynomial signature scheme. In: Ioannidis, J., Keromytis, A., Yung, M. (eds.) ACNS 05, LNCS, vol. 3531, pp. 164–175. Springer, Heidelberg (2005)
20. Ding, J., Xie, X., Lin, X.: A simple provably secure key exchange scheme based on the learning with errors problem. Cryptology ePrint Archive, Report 2012/688 (2012). http://eprint.iacr.org/2012/688
21. Ducas, L.: Accelerating bliss: the geometry of ternary polynomials. Cryptology ePrint Archive, Report 2014/874 (2014). http://eprint.iacr.org/2014/874
22. Ducas, L., Durmus, A., Lepoint, T., Lyubashevsky, V.: Lattice signatures and bimodal Gaussians. In: Canetti, R., Garay J.A. (eds.) CRYPTO 2013, Part I, LNCS, vol. 8042, pp. 40–56. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_3
23. Dwarakanath, N.C., Galbraith, S.D.: Sampling from discrete gaussians for lattice-based cryptography on a constrained device. Appl. Algebra Eng. Commun. Comput. **25**(3), 159–180 (2014)
24. Fujioka, A., Suzuki, K., Xagawa, K., Yoneyama, K.: Practical and post-quantum authenticated key exchange from one-way secure key encapsulation mechanism. In: Chen, K., Xie, Q., Qiu, W., Li, N., Tzeng, W.G. (eds.) ASIACCS 13, pp. 83–94. ACM Press, Hangzhou (2013)
25. Fujioka, A., Suzuki, K., Xagawa, K., Yoneyama, K.: Strongly secure authenticated key exchange from factoring, codes, and lattices. Des. Codes Cryptogr. **76**(3), 469–504 (2015)
26. Fujisaki, E., Okamoto, T.: How to enhance the security of public-key encryption at minimum cost. In: Imai, H., Zheng, Y. (eds.) PKC'99, LNCS, vol. 1560, pp. 53–68. Springer, Heidelberg (1999)
27. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Ladner, R.E., Dwork, C. (eds.) 40th ACM STOC, pp. 197–206. ACM Press, Victoria (2008)
28. Güneysu, T., Lyubashevsky, V., Pöppelmann, T.: Practical lattice-based cryptography: a signature scheme for embedded systems. In: Prouff, E., Schaumont, P. (eds.) CHES 2012, LNCS, vol. 7428, pp. 530–547. Springer, Heidelberg (2012)
29. Katz, J., Lindell, Y.: Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series). Chapman & Hall/CRC, Boca Raton (2007)
30. Krawczyk, H.: HMQV: a high-performance secure Diffie–Hellman protocol. In: Shoup, V. (ed.) CRYPTO 2005, LNCS, vol. 3621, pp. 546–566. Springer, Heidelberg (2005)
31. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) EUROCRYPT 2010, LNCS, vol. 6110, pp. 1–23. Springer, Heidelberg (2010)
32. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. J. ACM **60**(6), 43 (2013)
33. Lyubashevsky, V., Peikert, C., Regev, O.: A toolkit for ring-LWE cryptography. In: Advances in Cryptology—EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, 26–30 May 2013. Proceedings, pp. 35–54 (2013)

34. National Institute of Standards and Technology (NIST): Post-quantum cryptography: Nist's plan for the future (2015)
35. National Security Agency (NSA): Cryptography today. https://www.nsa.gov/ia/programs/suiteb_cryptography/ (2015)
36. Peikert, C.: An efficient and parallel Gaussian sampler for lattices. In: Advances in Cryptology—CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, 5–19 August 2010. Proceedings, pp. 80–97 (2010)
37. Peikert, C.: Lattice cryptography for the internet. In: Post-Quantum Cryptography—6th International Workshop, PQCrypto 2014, Waterloo, ON, Canada, 1–3 October 2014. Proceedings, pp. 197–219 (2014)
38. Peikert, C.: A decade of lattice cryptography. Found. Trends Theor. Comput. Sci. **10**(4), 283–424 (2016)
39. Petzoldt, A., Chen, M., Yang, B., Tao, C., Ding, J.: Design principles for HFEv- based multivariate signature schemes. In: Iwata, T., Cheon, J.H.(eds.) Advances in Cryptology—ASIACRYPT 2015—21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29–December 3, 2015, Proceedings, Part I, Lecture Notes in Computer Science, vol. 9452, pp. 311–334. Springer (2015). https://doi.org/10.1007/978-3-662-48797-6
40. Wolchok, S., Wustrow, E., Halderman, J.A., Prasad, H.K., Kankipati, A., Sakhamuri, S.K., Yagati, V., Gonggrijp, R.: Security analysis of India's electronic voting machines. In: Al-Shaer, E., Keromytis, A.D., Shmatikov, V. (eds.) ACM CCS 10, pp. 1–14. ACM Press, Chicago (2010)
41. Zhang, J., Zhang, Z., Ding, J., Snook, M., Dagdelen, Ö.: Authenticated key exchange from ideal lattices. In: Advances in Cryptology—EUROCRYPT 2015—34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, 26–30 April 2015, Proceedings, Part II, pp. 719–751 (2015)