Continuous Key Agreement with Reduced Bandwidth

Nir Drucker and Shay Gueron

University of Haifa, Israel,
and
Amazon, Seattle, USA

Abstract. Continuous Key Agreement (CKA) is a two-party procedure used by Double Ratchet protocols (e. g., Signal). This is a continuous and synchronous protocol that generates a fresh key for every sent/received message. It guarantees forward secrecy and post-compromise security. Alwen et al. have recently proposed a new KEM-based CKA construction where every message contains a ciphertext and a fresh public key. This can be made quantum-safe by deploying a quantum-safe KEM. They mention that the bandwidth can be reduced when using an ElGamal KEM (which is not quantum-safe). In this paper, we generalized their approach by defining a new primitive, namely Merged KEM (MKEM). This primitive merges the key generation and the encapsulation steps of a KEM. This is not possible for every KEM and we discuss cases where a KEM can be converted to an MKEM. One example is the quantum-safe proposal BIKE1, where the BIKE-MKEM saves 50% of the communication bandwidth, compared to the original construction. In addition, we offer the notion and two constructions for hybrid CKA.

Keywords: Double Ratchet Protocol · Continuous Key Agreement · Post Quantum Cryptography · Code-based Cryptography · BIKE

1 Introduction

Double Ratchet (DR) protocols (e. g., Signal [16]) are used to secure instant messaging applications such as WhatsApp [3], Skype [14], Facebook Messenger [1], and Google Allo [15]. Several formal security analyses of the DR protocol and its variants are given in [4, 6, 10, 11, 13, 17], focusing on slightly different sets of security properties. For example, according to [4] a secure DR based messaging protocol between parties A and B in the presence of an attacker $\mathcal{A}$ should have the following properties (see details in [4]):

- Correctness. If $\mathcal{A}$ is a passive attacker then A (B) receives all the messages sent by B (A) in the correct order.
- Immediate Decryption and Message-Loss Resilience. A message is decrypted upon arrival. In addition, the protocol execution continues even if a message is lost.

- Authenticity. $\mathcal{A}$ cannot modify messages or inject new ones (unless one of the parties' state is compromised).
- Privacy. $\mathcal{A}$ does not learn anything about the contents of the messages (unless one of the parties state is compromised).
- Forward Secrecy (FS). If one of the parties' state is compromised, the previous messages remain confidential.
- Post-Compromise Security (PCS). The parties can recover from a state compromise. Here, we assume that the parties have access to a randomness source and that $\mathcal{A}$ remains passive.
- Randomness leakage/failures. Fresh randomness is required only for achieving Post-Compromise Security (PCS) and is not used to achieve other property.

A DR protocol achieves these properties by: a) encrypting and authenticating every message with a fresh symmetric key; b) using fresh randomness (that is often used by some Public Key Encryption (PKE) scheme to achieve PCS).

In [4] the DR protocol uses a CKA (public-key ratchets), an AEAD with new keys for every message (symmetric-key ratchets), FS-AEAD hereafter, and a hash function. The CKA can be constructed from any PKE scheme, in particular, any IND-CPA KEM. The use of KEMs is also mentioned in [17] (although the construction in [4] is simpler). The DR scheme can be made quantum-safe[1] by using a quantum-safe KEM, FS-AEAD, and hash function.

To construct a CKA from a KEM, party A (wlog) uses a public key (received from B) to encapsulate a shared secret ss into a ciphertext ct, generates a new pair of secret/public keys (sk, pk) and sends ct and pk to B, at every round. This protocol doubles the communication bandwidth compared to DH CKA where only one public key is sent (assuming the DH and the KEM public keys of the same size). To reduce the bandwidth overhead, [4] mentions that deploying ElGamal KEM allows using the ciphertext as the subsequent public key. Note that ElGamal KEM is not quantum-safe.

Our contribution:

- We define a new primitive that we call MKEM. An MKEM is derived from a KEM by merging the key generation and the encapsulation procedures. Using it can achieve 50% bandwidth reduction compared to the original CKA protocol with the same KEM. It also may save some of the operations executed during encapsulation. We point out that converting a KEM to an MKEM is not always possible (e. g., BIKE2).

[1] We use the terminology quantum-safe to cryptographic algorithms that rely on problems that are believed to be hard even in the presence of quantum computers. For example, cryptographic algorithms that rely on factorization (e. g., RSA) are not considered quantum-safe due to Shor's algorithm [18]. On the other hand for some parameters cryptographic algorithms that rely on the Shortest Vector Problem over lattices are commonly considered quantum-safe.

- We present an instantiation of MKEM with BIKE1 [5].
- Following Bindel et al. [8] (who introduced compilers for quantum-safe hybrid Authenticated Key Exchange) we propose two compilers for a hybrid CKA. We believe that this is the first hybrid quantum-safe CKA compiler.

The organization of the paper. Section 2 introduces some notation and background. In Section 3 we describe the MKEM scheme and its properties. We present an MKEM instantiation that is based on BIKE1 in Section 4, and two hybrid CKA constructions in Section 5. Section 6 is the conclusion.

2 Notation and background

We denote null values and protocol failures by $\perp$. Uniform random sampling from a set W is denoted by $w \xleftarrow{\$} W$. For an algorithm A, we denote its output by $out = A()$ if A is deterministic, and by $out \leftarrow A()$ otherwise. The (Hamming) weight of a vector of bits x is denoted by $wt(x)$. The finite field of 2 elements is denoted by $\mathbb{F}_2$. The xor operation is denoted by $\oplus$. An attacker $\mathcal{A}$ is parameterized by its running time t. Let the term epoch denote the period between two consecutive messages sent by the same party in a DR protocol. During an epoch, the other party can send as many messages as it wishes.

2.1 Continuous Key Agreement (CKA)

A CKA (roughly) models the public-key ratchets in a DR protocol. It can use a PKE of choice and in particular a KEM. This synchronous protocol between parties A and B sends a message msg_i in round i: from A to B if i is even and from B to A otherwise. A fresh shared secret ss_i is agreed by both parties in round i. The state of the parties is denoted by $\gamma^{(A)}$ and $\gamma^{(B)}$, respectively.

Definition 1. *A CKA scheme consists of four algorithms CKA =* (CKA-Init-A, CKA-Init-B, CKA-S, CKA-R)*, where*

- CKA-Init-A *(*CKA-Init-B*) gets an input key k and outputs an initial state. The notation is* $\gamma^{(A)} \leftarrow$CKA-Init-A(k) ($\gamma^{(B)} \leftarrow$ CKA-Init-B(k))*, respectively.*
- CKA-S *updates the party's state* $\gamma^{(\cdot)}$*, generates a message* msg_i*, and a key* ss_i*. The notation is:* $(\gamma^{(\cdot)'}, msg_i, ss_i) \leftarrow$ CKA-S$(\gamma^{(\cdot)})$.
- CKA-R *for an input message* msg_i *and a state* $\gamma^{(\cdot)}$ *generates a new state* $\gamma^{(\cdot)'}$ *and calculates the shared secret. The notation is* $(\gamma^{(\cdot)'}, ss_i) \leftarrow$ CKA-R$(\gamma^{(\cdot)}, msg_i)$.

CKA-S$(\gamma^{(\cdot)})$ *is a randomized algorithm. For adversarial cases where the source of randomness is controlled by an adversary, we denote the algorithm by* CKA-S$(\gamma^{(\cdot)}, r)$*. Here r denotes the adversary controlled randomness. We use $\mathcal{K}$ to denote the set of initial keys and $\mathcal{SS}$ to denote the set of possible shared secrets* ss_i*,* $i = 1, 2, \ldots$.

We briefly describe the CKA correctness property and its security game (a full description is found in [4]).

Correctness. A CKA scheme is correct if for every $i = 1, 2, \ldots$, A and B agree (with high probability) on the same (shared) secret ss_i.

Security. A challenger Chal sets the epoch counters $t_A = t_B = 0$, samples a bit $b \xleftarrow{\$} \{0,1\}$, an initial key $k \xleftarrow{\$} \mathcal{K}$, and invokes $\gamma^{(A)} \leftarrow \mathsf{CKA\text{-}Init\text{-}A}(k)$ and $\gamma^{(B)} \leftarrow \mathsf{CKA\text{-}Init\text{-}B}(k)$. Chal receives an input t^* that defines the round on which the challenge oracle may be called.
Let U denote one of the users A or B. An adversary $\mathcal{A}$ interacts with Chal by making oracle calls to one of the following five oracles (in a ping-pong order $A \to B \to A \to \ldots$):

1. $\mathsf{Send\text{-}U}()$: increment t_U by 1, perform $(\gamma^U, msg_{t_U}, ss_{t_U}) \leftarrow \mathsf{CKA\text{-}S}(\gamma^U)$, and return (msg_{t_U}, ss_{t_U}).
2. $\mathsf{Send\text{-}U'}(r)$: increment t_U by 1, perform $(\gamma^U, msg_{t_U}, ss_{t_U}) \leftarrow \mathsf{CKA\text{-}S}(\gamma^U, r)$, and return (msg_{t_U}, ss_{t_U}). This oracle can be called only if $\max(t_A, t_B) \leq t^* - 2$.
3. $\mathsf{Receive\text{-}U}()$: increment t_U by 1 and perform $(\gamma^U, \cdot) = \mathsf{CKA\text{-}R}(\gamma^U, msg_{t_U})$.
4. $\mathsf{Corr\text{-}U}()$: return γ^U. A call to this oracle is allowed only when $\max(t_A, t_B) \leq t^* - 2$ or $t_U \geq t^* + \Delta_{CKA}$. ($\Delta_{CKA}$ is defined below).
5. $\mathsf{Chall\text{-}U}()$: increment t_U by 1, and perform $(\gamma^U, msg_{t_U}, ss_{t_U}) \leftarrow \mathsf{CKA\text{-}S}(\gamma^U)$. If $b = 0$ return (msg_{t_U}, ss_{t_U}), and otherwise set $ss' \xleftarrow{\$} \mathcal{SS}$ and return (msg_{t_U}, ss). This oracle can be invoked only when $t_U = t^*$ and if no Corr() or Send-U'() calls were performed less than two epochs before the call.

The game is parameterized by Δ_{CKA}: the minimum number of epochs between t^* and a state that do not contain secrets. When a party reaches epoch $t^* + \Delta_{CKA}$, its state may be revealed to $\mathcal{A}$ (by a Corr-U() call). The game is endless but we consider it terminated if $\gamma^{(A)}$ and $\gamma^{(B)}$ are revealed or when $\mathcal{A}$ outputs a bit b'. $\mathcal{A}$ wins if $b' = b$. The advantage of $\mathcal{A}$ against a CKA with $\Delta_{CKA} = \Delta$ is denoted by $Adv^{CKA}_{ror,\Delta}(\mathcal{A})$.

Definition 2. *A CKA scheme is (t, Δ, ϵ)-secure if for all t-attackers $\mathcal{A}$,*

$$Adv^{CKA}_{ror,\Delta}(\mathcal{A}) \leq \epsilon \tag{1}$$

2.2 Key Encapsulation Mechanism (KEM)

A KEM is a public key primitive. We denote the secret key and public key domains by $\mathcal{SK}$ and $\mathcal{PK}$, respectively. A KEM consists of three functions: $\mathtt{keygen}$, encaps, decaps. It is played between parties A and B through three messages (sent over an un-trusted channel). First, A invokes $(sk, pk) \leftarrow \mathtt{keygen}(1^\kappa)$ generating a secret key $sk \in \mathcal{SK}$ and a public key $pk \in \mathcal{PK}$, and sends pk to B. B uses the received pk and invokes $(ss, ct) \leftarrow \mathsf{encaps}(1^\kappa, pk)$ to produce a ciphertext ct and a shared secret $ss \in \mathcal{SS}$. B sends ct to A. A uses the received ct and invokes $ss = \mathsf{decaps}(sk, ct)$ (in some KEM protocols decaps may occasionally fail. In such cases we say that the output is $\perp$).

3 Merged KEM (MKEM)

We propose MKEM as a primitive for CKA.

Definition 3. *An MKEM is a public-key primitive with two algorithms* $MKEM = (\mathsf{kgc}, \mathsf{decaps})$ *that have the following syntax:*

- kgc. *Take an (implicit) security parameter and a public key* pk_0 *and output* (sk_1, pk_1, ct_1, ss_1). *Here,* (sk_1, pk_1) *is a newly generated key pair. If* $pk_0 = \perp$ *then* $ct_1 = ss_1 = \perp$ *(i. e., output* $(sk_1, pk_1, \perp, \perp) \leftarrow \mathsf{kgc}(\perp)$*). Otherwise, use* pk_0 *to generate a ciphertext* ct_1*, in a way that* pk_1 *and a shared secret* ss_1 *can be retrieved from* ct_1 *by invoking* decaps.
- decaps*: receive a secret key* sk_0 *and a ciphertext* ct_1 *and retrieve the shared secret* ss_1 *and* pk_1*, i. e.,* $(ss_1, pk_1) = \mathsf{decaps}(sk_0, ct_1)$.

Remark 1. In MKEM, only the initial public key pk_0 is non-secret. For $i \geq 1$, pk_{i-1} and pk_i have no use after calling $(\cdot, pk_i, \cdot, \cdot) \leftarrow \mathsf{kgc}(pk_{i-1})$, and can be deleted immediately after this invocation.

Correctness. Consider the (continuous) iterative sequences: A executes $(sk_0, pk_0, \perp, \perp) \leftarrow \mathsf{kgc}(\perp)$ and sends pk_0 to B; B executes $(sk_1, pk_1^B, ct_1, ss_1^B) \leftarrow \mathsf{kgc}(pk_0)$ and sends ct_1 to A; A repeat the process by executing $(ss_1^A, pk_1^A) = \mathsf{decaps}(sk_0, ct_1)$ and $(sk_2, pk_2^B, ct_2, ss_2^B) \leftarrow \mathsf{kgc}(pk_1)$ and sending ct_2 to B and so on. We say that an MKEM is correct if for each $i \geq 1$, $ss_i^A = ss_i^B$ and $pk_i^A = pk_i^B$.

Security. The security of an MKEM is defined similarly to the IND-CPA and IND-CCA security of a KEM. Let Chal be the game challenger and let $\mathcal{A}$ be an adversary.

- IND-CPA: Chal generates $(sk_1, pk_1, ct_1 = \perp, ss_1 = \perp) \leftarrow \mathsf{kgc}(\perp)$ or $(sk_1, pk_1, ct_1, ss_1) \leftarrow \mathsf{kgc}(pk_0)$, $pk_0 \in \mathcal{PK}$, computes $(\cdot, \cdot, ct_2, ss_2^0) \leftarrow \mathsf{kgc}(pk_1)$, and samples $ss_2^1 \xleftarrow{\$} \mathcal{SS}$, $b \xleftarrow{\$} \{0, 1\}$. It hands (ct_1, ct_2, ss_2^b) to $\mathcal{A}$ that outputs a bit b' (indicating whether it believes it received ss_2^0 or ss_2^1). $\mathcal{A}$ wins if $b' = b$.
- IND-CCA - Here, $\mathcal{A}$ also has access to a decaps oracle. This oracle returns $(\hat{ss}, \hat{pk})$ for every input $\hat{ct} \neq ct_1, ct_2$.

Definition 4. *An MKEM scheme is* (t, ϵ)*-cpa-secure if for all* t*-attacker* $\mathcal{A}$*,*

$$Adv_{cpa}^{MKEM}(\mathcal{A}) \leq \epsilon \tag{2}$$

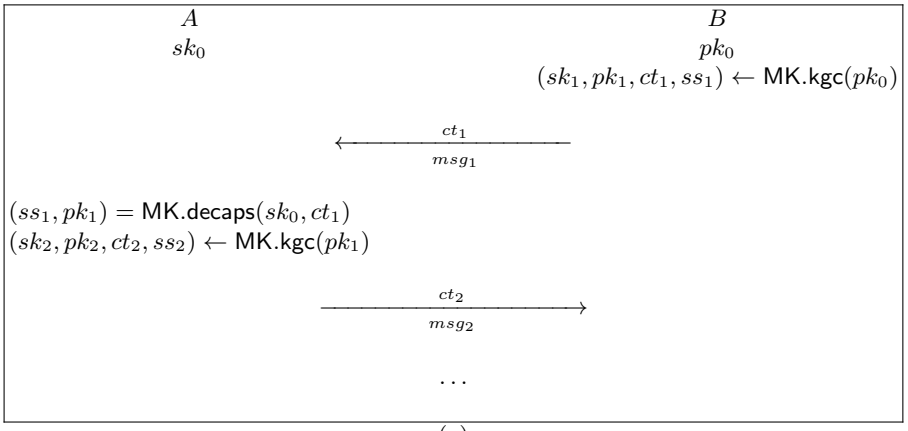
Figure 1 shows the flow of a CKA that uses an MKEM (Panel (a)) and also compares (Panel (b)) to a CKA based on a KEM. We require that the MKEM is IND-CPA (similarly to KEM, IND-CCA is not required).

Constructing an MKEM scheme is not necessarily simple. Indeed, in Section 4 we show how to transform BIKE1 KEM into BIKE1-MKEM and explain why the same technique cannot be applied to BIKE2/3. Consequently, we work on each case separately, without stating a general security relation between an MKEM and its related KEM (although we believe that equivalence exists).

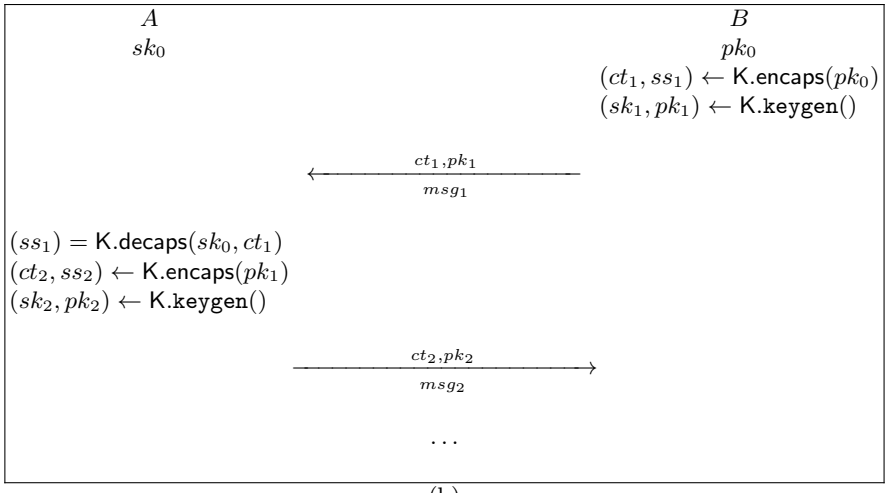
Lemma 1. *Let* MK *be a* (t', ϵ)*-cpa-secure MKEM. Then, the corresponding CKA scheme (denoted* CKA*) is* $(t, \Delta = 0, \epsilon)$*-secure for* $t \approx t'$*.*

Proof (outline). According to [4] (see Theorem 2 therein): for every KEM K that is (t'', ϵ)-cpa-secure ($t'' \approx t$), there is an adversary $\mathcal{B}$ for which

$$Adv^{CKA}_{ror,0}(\mathcal{A}) \leq Adv^{KEM}_{cpa}(\mathcal{B}) \tag{3}$$



(a)



(b)

Fig. 1: Panel a: A CKA protocol that uses an MKEM (MK). Panel B: A CKA protocol that uses KEM (K). The initialization $sk_0 \leftarrow$CKA-Init-A(k), $pk_0 \leftarrow$CKA-Init-A(k) (not shown in the figures) starts A with sk_0 and B with pk_0. At each subsequent round (i) the new shared secret (ss_i) is generated.

Replacing the KEM with the analogous MKEM does not change the confidentiality of the messages that $\mathcal{A}$ can see (it sees a ciphertext in both cases).

Epoch t_A starts when A sends msg_{2t_A-1} and ends when it sends msg_{2t_A+1}. If in this epoch, $\mathcal{A}$ performs Corr-A(), it gets to see $\gamma^A = sk_{2t_A-1}$ (or $\gamma^A = (sk_{2t_A+1})$ if msg_{2t_A} was already received). This allows $\mathcal{A}$ to decapsulate ct_{2t_A} (resp. ct_{2t_A+2}) and extract (ss_{2t_A}, pk_{2t_A}) (resp. $(ss_{2t_A+2}, pk_{2t_A+2})$). It provides no information on sk_{2t_A} $(resp. ss_{2t_A+2})$ to $\mathcal{A}$, by the properties of the underlying KEM. Consequently, $\Delta = 0$ also when using MKEM. □

Remark 2. In the proof of Lemma 1 if $\mathcal{A}$ gets to see some pk_i, $i \geq 1$ value it may be able to decrypt/decapsulate ct_i (that was derived from pk_i) and obtain ss_i. However by the properties of the underlying KEM, $\mathcal{A}$ cannot obtain sk_i.

4 BIKE-MKEM

BIKE [5] is a suite of 3 KEMs (BIKE1, BIKE2, BIKE3) submitted to the NIST Post-Quantum Cryptography (PQC) project ([2]). They are IND-CPA secure KEMs. BIKE1/2/3 use Quasi Cyclic - Moderate Density Parity Check (MDPC) codes, to enjoy shorter keys than McEliece KEM [7]. Fig. 2 outlines BIKE1/2/3, and full details are available in [5].

The computations of BIKE are executed over $\mathcal{R} := \mathbb{F}_2[X]/\langle X^r - 1\rangle$, for the parameter r. Denote the (Hamming) weights of the secret key $sk = (sk_0, sk_1)$ and the errors vector e by w and t, respectively. Concrete BIKE1 parameters for NIST Level-1 are $|pk| = |ct| = 20,326$, $|ss| = 256$ $r = 10,163$, $w = 142$, and $t = 134$. BIKE1/2/3 are IND-CPA KEM because decoding failures may occur, at some low rate, estimated to be at most 10^{-8}. Therefore, and also to achieve forward secrecy, BIKE1/2/3 use ephemeral keys.

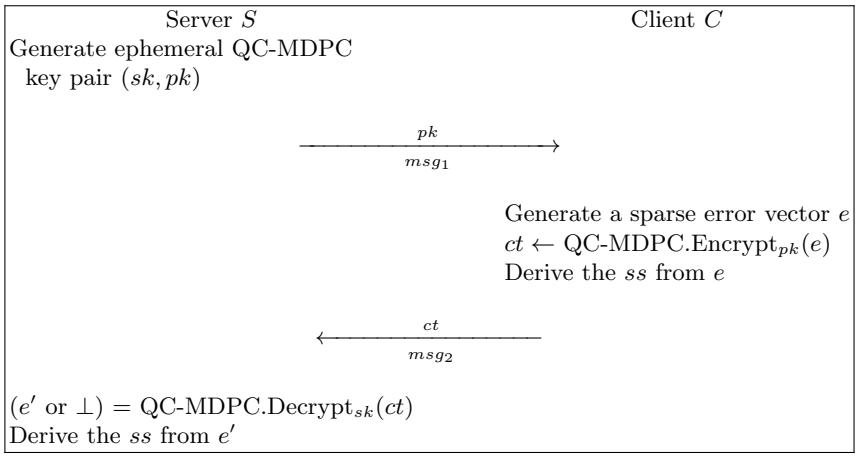


Fig. 2: A general description of BIKE1/2/3 protocol.

4.1 BIKE1-MKEM transformation

Figure 3 shows the proposed BIKE1-MKEM. For every $X \in \{pk, sk, ct, e\}$,$i = \{0,1\}$ X consists of two equal length halves $(X[0], X[1])$ (e. g., $pk = (pk[0], pk[1])$). We explain the elements of the protocol below.

- $\mathsf{kgc}(pk_0)$. Receive $pk_0 \in \mathcal{PK}$ as input. Generate a secret key sk_1 with odd weights of $\approx w/2$ (for $sk_1[0]$ and for $sk_1[1]$). Generate $g \xleftarrow{\$} \mathcal{R}$ and calculate the public key $pk_1 = (g \cdot sk_1[1],\ g \cdot sk_1[0])$. Sample $b_0, b_1 \xleftarrow{\$} \{0,1\}$ and set $pk'_1 = (pk_1[0] \oplus b_0, pk_1[1] \oplus b_1)$ (see Remark 4 for the requirement on pk'_1). Subsequently, generate an error vector $e \xleftarrow{\$} \mathcal{R}$ with weight t and use pk'_1 and pk_0 to encrypt it to a ciphertext $ct = (pk_0[0] \cdot pk'_1[0] + e_0,\ pk_0[1] \cdot pk'_1[1] + e_1)$. Hash the error vector e to generate the value ss (which is the shared secret). Output sk, pk_1, ct, and ss.
- $\mathsf{decaps}(sk_0, ct)$. Receive $sk_0 \in \mathcal{SK}$ as input. Compute the syndrome $synd = ct[0] \cdot sk_0[0] + ct[1] \cdot sk_0[0]$ and decode $synd$ (with a QC-MDPC decoder) to extract the error vector e'. If the decoding succeeds and also $wt(e') = t$, calculate

$$\begin{aligned} pk'_1 &= (pk'_1[0], pk'_1[1]) \\ &= \left(pk_0^{-1}[0]\left(ct[0] - e'[0]\right),\ pk_0^{-1}[1]\left(ct[1] - e'[1]\right)\right) \end{aligned}$$

  (see Remark 5 for how to calculate $pk_0^{-1}[0]$ and $pk_0^{-1}[1]$). If $pk'_1[i]$, $i = 0, 1$ is even set $pk'_1[i] = pk'_1[i] \oplus 1$. Set $pk_1 = (pk'_1[0], pk'_1[1])$ and derive the shared secret ss by hashing e'.

Remark 3. The encapsulation in BIKE1 (which is a KEM) samples a random message $m \xleftarrow{\$} \{0,1\}^n$. The decapsulation needs only to retrieve the error vector but not m itself. In BIKE1-MKEM the decapsulation needs to extract both the shared secret ss and the public key pk_1.

Remark 4. In BIKE1 MKEM we replace m with $pk_1 = (pk_1[0], pk_1[1]) \in \mathcal{PK} = \{0,1\}^n$`-with-even-weight`. Thus, we need to convert it to be a uniform random element in $\{0,1\}^n$. To this end, we sample two bits $b_i \xleftarrow{\$} \{0,1\}$, $i = 0, 1$, and xor them to the least significant bit of $pk_1[i]$. During decapsulation (after extracting pk'_1), decaps checks if one of its halves has even weight, and flips its least significant bit accordingly.

Remark 5. The values $pk_0^{-1}[0], pk_0^{-1}[1]$, required to retrieve pk_1, can be calculated during either kgc or decaps (they are invertible by the definition of BIKE1). In the first case we extend the "structure" of the secret key to $sk_1 = (sk_1[0], sk_1[1], pk_0^{-1}[0], pk_0^{-1}[1]$ and in the second case we change it into $sk_1 = (sk_1[0], sk_1[1], pk_0[0], pk_0[1])$, respectively.

Correctness. The correctness of BIKE1-MKEM follows by inspecting the flows, up to possible decoding failures that, for BIKE1-MKEM occur at a Decoding Failure Rate (DFR) $\leq 10^{-8}$.

$(\mathbf{sk_1}, \mathbf{pk_1}, \mathbf{ct}, \mathbf{ss}) \leftarrow \mathsf{kgc}(\mathbf{pk_0})$	$(\mathbf{ss}, \mathbf{pk}) = \mathsf{decaps}(\mathbf{sk_0}, \mathbf{ct})$
$sk_1 \xleftarrow{\$} \mathcal{R}$, $wt(sk_1[0])$, $wt(sk_1[1])$ is odd and $\approx w/2$ $g \xleftarrow{\$} \mathcal{R}$ of odd weight $pk = (g \cdot sk_1[1], g \cdot sk_1[0])$	$synd = ct[0] \cdot sk_0[0] + ct[1] \cdot sk_0[1]$ $e' = \text{BIKE.decode}(synd, sk')$ Abort if $(e' = \perp)$ Abort if $wt(e') \neq t$
$b_0, b_1 \xleftarrow{\$} \{0,1\}$ $pk'_1 = (pk_1[0] \oplus b_0, pk_1[1] \oplus b_1)$ $e \xleftarrow{\$} \mathcal{R}^2$ such that $wt(e[0]) + wt(e[1]) = t$ $ct = (pk_0[0] \cdot pk'_1[0] + e[0],$ $\quad pk_0[1] \cdot pk'_1[1] + e[1])$	$pk'_1 = (pk_0^{-1}[0](ct[0] - e'[0]),$ $\quad pk_0^{-1}[1](ct[1] - e'[1]))$ If $wt(pk'_1[0])$ is even $pk'_1[0] = pk'_1[0] \oplus 1$ If $wt(pk'_1[1])$ is even $pk'_1[1] = pk'_1[1] \oplus 1$ $pk_1 = (pk'_1[0], pk'_1[1])$
$ss = \texttt{Parallel\_SHA}^{384}(e)$	$ss = \texttt{Parallel\_SHA}^{384}(e')$

Fig. 3: BIKE1 MKEM. Here, note that for every $X \in \{pk_0, sk_0, pk_1, sk_1, e, ct\}$, X consists of two equal length halves $(X[0], X[1])$ (e. g., $ct = (ct[0], ct[1])$). $\texttt{Parallel\_SHA}^{384}$ is the hash function (that was optimized for performance) used by BIKE [5].

Lemma 2. *Let BIKE1-KEM be a (t, ϵ)-cpa-secure KEM. Then, BIKE1-MKEM is a (t, ϵ)-cpa-secure MKEM.*

Proof. Let $\mathcal{A}$ be an adversary against BIKE1-MKEM. We construct an adversary $\mathcal{B}$ against the IND-CPA property of BIKE1. $\mathcal{B}$ receives a triple (pk, ct, ss_b) and attempts to guess $b = \{0, 1\}$ as described before. It hands (pk, ct, ss_b) to $\mathcal{A}$ and outputs the same bit that $\mathcal{A}$ outputs. $\mathcal{A}$ cannot distinguish a ciphertext that was generated by BIKE1-KEM from a ciphertext generated by BIKE1-MKEM, because the generation is equivalent. Therefore,

$$Adv_{cpa}^{BIKE1-MKEM}(\mathcal{A}) \leq Adv_{cpa}^{BIKE1-KEM}(\mathcal{B}) \leq \epsilon \tag{4}$$

(we consider the same t for both $\mathcal{A}$ and $\mathcal{B}$). □

4.2 CKA, MKEM and DFR

This section discusses the difficulties that arise from using a KEM/MKEM that has non-negligible DFR (e. g., BIKE1) for constructing a CKA[2]. Consider, the case in the DR protocol of [4], where (wlog) A sends a $msg_i = (ct, pk)$ to B, and B cannot decapsulate it (due to a decapsulation error). In this case the DR protocol stalls: B ignores msg_i and leaves its epoch counter t_B unchanged. A

[2] CKA uses ephemeral keys for both KEM and MKEM. This protects the scheme from attacks that may exploit decapsulation failures, such as [12] in the context of QC-MDPC codes. We note that CKA is aborted (and subsequently re-initialized) upon encountering a decapsulation failure.

that does not expect an acknowledgement, continues to use the "bad" ciphertext ct for its subsequent messages, during the epoch that has "no reason" to change. The motivation for not sending an acknowledgement in response to msg_i is: a) the DR protocol is asynchronous; b) avoid a Denial of Service (DoS) situation that occurs when $\mathcal{A}$ deliberately sends "bad" messages to B that cannot be decapsulated. Here, sending (a failure) "acknowledgement" would overload the network.

The DFR of BIKE1-MKEM is at most 10^{-8}. We argue that this can be tolerated from the practical viewpoint. Consider a user that performs $10,000$ conversations, using $10,000$ epochs. Every epoch includes at least one message. Even in this extreme scenario, the user is expected to experience a decoding failure at most once. From the practical viewpoint, it means that BIKE1-MKEM is correct 99.999999% of the time.

A general treatment for DFRs in DR protocols is left as a future work, but we provide here, some practical remedies.

1. A messaging application can offer "refresh"/"restart" functionality as commonly done in many applications. When a user expects messages but notices that none arrive for a "long" period of time he can invoke a "restart"/"refresh" for the conversation. This alleviates inconvenience inflicted by decoding errors. Stalls due to DoS attacks are captured in [4].
2. A messaging application can use a timer. If no response arrives after a long period of time the application can automatically restart the connection.
3. A receiver who fails to decapsulate a message can alert the sender. This approach is not ideal because it can lead to a DoS attack. Unless, the receiver can distinguish between benign decapsulation errors and maliciously-sent "bad" messages. An example for such case is the Public Key Secure-Messaging (PKSM) of [4].

5 A Hybrid CKA constructions

Currently, new standards for quantum-safe key exchange, encryption and signatures are developed, but no finalize vetted schemes are available for immediate deployment (the NIST process [2] is expected to last a few more years). However, threats (at least theoretical) to current CKAs exist: recorded sessions that are secure in the classical world may be broken in a Post-Quantum (PQ) setting. A hybrid approach that combines a classical and a quantum-safe scheme seems to be a prudent approach, hoping to achieve post-quantum security without taking the risk of a premature transition to an un-vetted scheme. To this end, some hybrid Key Exchange (KEX) protocols and combiners have been recently suggested (constructions and useful survey are given in [8]).

We extend the list of hybrid KEX/AKE/SSH with a new notion, of a Hybrid CKA (H-CKA). Concretely, we propose two constructions. Parallel H-CKA and Interleaved H-CKA, both using the hybrid KEM of [8].

Parallel H-CKA. This is a combination of two CKA protocols: classical CKA^c and quantum-safe CKA^q (as illustrated in Fig. 4). Here, $\gamma_1^A = (sk^c, sk^q) \leftarrow$CKA-Init-A$(k^c, k^q)$ and $\gamma_2^B = (pk^c, pk^q) \leftarrow$CKA-Init-B$(k^c, k^q)$, where $k^c, k^q \in \mathcal{K}$, $pk^c \in \mathcal{PK}^c$, $pk^q \in \mathcal{PK}^q$ and sk^c, sk^q are the associated secret keys.

- The procedure $\left(\gamma', msg_i = (ct_i^c, ct_i^q), ss_i\right) \leftarrow$CKA-S$(\gamma)$: 1) calculate (in parallel)

$$(sk_i^c, pk_i^c, ct_i^c, ss_i^c) \leftarrow \mathsf{MK}^c.\mathsf{kgc}(pk_{i-1}^c)$$
$$(sk_i^q, pk_i^q, ct_i^q, ss_i^q) \leftarrow \mathsf{MK}^q.\mathsf{kgc}(pk_{i-1}^q)$$

  2) apply a combiner (e. g., as in [8]) $ss_i = \texttt{combine}(ss_i^c, ss_i^q, msg_i)$ and generate the shared secret ss_i; 3) set $\gamma' = (sk_i^c, sk_i^q)$.
- The procedure $\left(\gamma', ss_i\right) \leftarrow$CKA-R$(\gamma, msg_i)$: 1) decapsulate (ct_i^c, ct_i^q) to extract $(ss_i^c, ss_i^q, pk_i^c, pk_i^q)$; 2) set $\gamma = (pk_i^c, pk_i^q)$ and apply the same combiner as above.

There are no additional (sub)rounds in the Parallel H-CKA compared to CKA. However, the communication bandwidth is the sum of the bandwidths of the two involved schemes. We note that the H-CKA construction uses MKEMs, but it also possible to use KEMs instead.

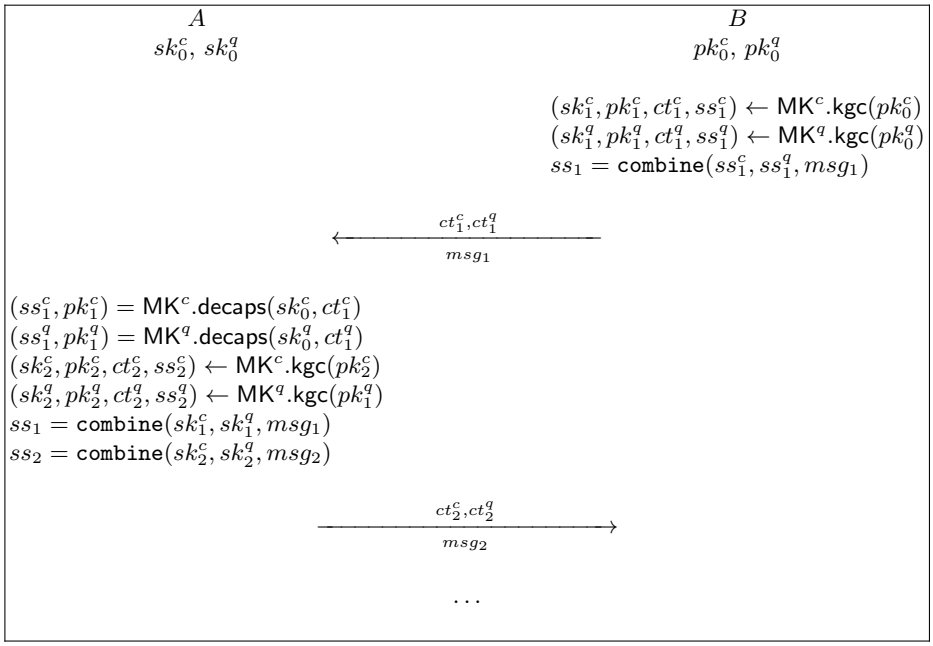


Fig. 4: Parallel Hybrid CKA (H-CKA) combining a classicical security and quantum-safe MKEMs (MK^c and MK^q, respectively). The combiner `combine` is one of the options of [8].

Interleaved CKA. An Interleaved CKA uses a CKA that is $(t, 2\Delta, \epsilon)$-secure instead of (t, Δ, ϵ)-secure. This means that recovering from a state compromise takes 2Δ rounds rather than only Δ. By [4], when a CKA uses KEM, we have $\Delta = 0$ (for comparison note that using DH has $\Delta = 2$). Therefore, our interleaved schemes are at least $(t, 1, \epsilon)$-secure. Instantiating an interleaved H-CKA can be done in two ways (deploying a "third ratchet").

1. Option 1. We break Parallel H-CKA into two interleaved flows: 1) $B \to A : ct_1^c$; 2) $A \to B : ct_1^q$ 3) $B \to A : ct_2^c$; 4) $A \to B : ct_2^q$. The sequence is repeated. The associated shared secrets of each round are ss_1^c, ss_1^q, ss_2^c, ss_2^q, ... In an odd round number i, $ss_i = \mathtt{combine}(ss_{i/2}^c, ss_{i/2}^q)$. In an even round i, $ss_i = \mathtt{combine}(ss_{i/2-1}^q, ss_{i/2}^c)$.
2. Option 2. We send the same messages msg_i as in Parallel H-CKA, but adding a Boolean toggle flag f to γ, where: if $\mathsf{f} = \mathsf{true}$ CKA-S and CKA-R operate as before but $ss_i = \mathtt{combine}(ct_{i-1}^c, ct_i^q)$; if $\mathsf{f} = \mathsf{false}$ no message will be sent/received and $ss_i = \mathtt{combine}(ct_i^c, ct_i^q)$.

The bandwidth in Option 1 is reduced (hopefully, by 50%) compared to Parallel H-CKA. The bandwidth in Option 2 is the same as in Parallel H-CKA but the number of rounds is halved. The tradeoff implied by using Interleaved H-CKA is that Δ is increased by at least 1. This can be tolerated for achieving better bandwidth/latency compared to Parallel H-CKA.

6 Conclusion

The new primitive MKEM is designed to reduce the bandwidth of the CKA protocol used by the DR scheme. It can also be used by any cryptographic protocol that uses two (or more) back to back KEMs. A concrete instantiation that is based on BIKE1, shows that it can have a significant impact (of 50%) on the bandwidth.

We are not sure if every KEM can be converted into an MKEM and if the bandwidth reduction is necessarily significant. Here are two examples.

- Many KEMs are based on a PKE scheme where the encryption is designed to operate on "short" messages. Consider Kyber512 [9] for instance. Its public key has 736 bytes, its ciphertext has 800 bytes, and it encrypts a 32-byte random message. Here, applying our MKEM method is easy (replacing one randomized value with another). However, this will reduce the bandwidth from $800 + 736 = 1,536$ to $800 + 704 = 1,504$ bytes i. e., save $32/1536 \approx 2\%$. It is still worth doing (at practically no cost), but the impact is modest.
- Consider BIKE2/3 that encrypt an error vector. This error vector has a specific weight that is different from the weight of the public key. Here, encrypting the public key (instead of the error vector) requires some transformation from different sets of bit strings (that have different cardinalities).

We suggest that H-CKA (Parallel H-CKA and Interleaved H-CKA) can be used by messaging applications to hopefully achieve quantum-safe security but without giving up the classical security.

We raised the difficulty of designing a general CKA primitive and DR scheme (beyond the practical proposed remedies) that can use KEMs that have a non-negligible DFR. This is left as an open problem.

Acknowledgments

This research was supported by: The Israel Science Foundation (grant No. 1018/16); The BIU Center for Research in Applied Cryptography and Cyber Security, in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office; the Center for Cyber Law & Policy at the University of Haifa in conjunction with the Israel National Cyber Directorate in the Prime Ministers Office.

References

1. −: Messenger secret conversations: Technical whitepaper. https://fbnewsroomus.files.wordpress.com/2016/07/secret_conversations_whitepaper-1.pdf (2013)
2. −: Nist:post-quantum cryptography - call for proposals (September 2017), https://csrc.nist.gov/Projects/Post-Quantum-Cryptography
3. −: Whatsapp encryption overview: Technical white paper. https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf (December 2017)
4. Alwen, J., Coretti, S., Dodis, Y.: The Double Ratchet: Security Notions, Proofs, and Modularization for the Signal Protocol. Cryptology ePrint Archive, Report 2018/1037 (2018), https://eprint.iacr.org/2018/1037
5. Aragon, N., Barreto, P., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.C., Gaborit, P., Gueron, S., Guneysu, T., Melchor, C.A., et al.: BIKE: Bit Flipping Key Encapsulation (2017)
6. Bellare, M., Singh, A.C., Jaeger, J., Nyayapati, M., Stepanovs, I.: Ratcheted Encryption and Key Exchange: The Security of Messaging. In: Katz, Jonathan and Shacham, Hovav (ed.) Advances in Cryptology – CRYPTO 2017. pp. 619–650. Springer International Publishing, Cham (2017)
7. Bernstein, D.J., Chou, T., Lange, T., Maurich, I.v., Misoczki, R., Niederhagen, R., Persichetti, E., Peters, C., Schwabe, P., Sendrier, N., Szefer, J., Wang, W.: Classic McEliece: conservative code-based cryptography (2017)
8. Bindel, N., Brendel, J., Fischlin, M., Goncalves, B., Stebila, D.: Hybrid key encapsulation mechanisms and authenticated key exchange. Cryptology ePrint Archive, Report 2018/903 (September 2018), http://eprint.iacr.org/
9. Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehl, D.: CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM. Cryptology ePrint Archive, Report 2017/634 (2017), https://eprint.iacr.org/2017/634

10. Cohn-Gordon, K., Cremers, C., Dowling, B., Garratt, L., Stebila, D.: A Formal Security Analysis of the Signal Messaging Protocol. In: 2017 IEEE European Symposium on Security and Privacy (EuroS P). pp. 451–466 (April 2017). https://doi.org/10.1109/EuroSP.2017.27
11. Durak, F.B., Vaudenay, S.: Bidirectional Asynchronous Ratcheted Key Agreement without Key-Update Primitives. Cryptology ePrint Archive, Report 2018/889 (2018), https://eprint.iacr.org/2018/889
12. Guo, Q., Johansson, T., Stankovski, P.: A Key Recovery Attack on MDPC with CCA Security Using Decoding Errors, pp. 789–815. Springer Berlin Heidelberg, Berlin, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_29
13. Jaeger, J., Stepanovs, I.: Optimal Channel Security Against Fine-Grained State Compromise: The Safety of Messaging. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology – CRYPTO 2018. pp. 33–62. Springer International Publishing, Cham (2018)
14. Lund, J.: Signal partners with microsoft to bring end-to-end encryption to skype. https://signal.org/blog/skype-partnership (October 2018)
15. Marlinspike, M.: Open whisper systems partners with google on end-to-end encryption for allo. https://signal.org/blog/allo/ (2013)
16. Perrin, T., Marlinspike, M.: The double ratchet algorithm. GitHub wiki (2016)
17. Poettering, B., Rsler, P.: Asynchronous ratcheted key exchange. Cryptology ePrint Archive, Report 2018/296 (2018), https://eprint.iacr.org/2018/296
18. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th Annual Symposium on Foundations of Computer Science. pp. 124–134 (Nov 1994). https://doi.org/10.1109/SFCS.1994.365700