A Public-Key Encryption Scheme with Pseudo-random Ciphertexts

Bodo Möller*

University of California, Berkeley
bmoeller@eecs.berkeley.edu

Abstract. This work presents a practical public-key encryption scheme that offers security under adaptive chosen-ciphertext attack (CCA) and has pseudo-random ciphertexts, i.e. ciphertexts indistinguishable from random bit strings. Ciphertext pseudo-randomness has applications in steganography. The new scheme features short ciphertexts due to the use of elliptic curve cryptography, with ciphertext pseudo-randomness achieved through a new key encapsulation mechanism (KEM) based on elliptic curve Diffie-Hellman with a pair of elliptic curves where each curve is a twist of the other. The public-key encryption scheme resembles the hybrid DHIES construction; besides by using the new KEM, it differs from DHIES in that it uses an authenticate-then-encrypt (AtE) rather than encrypt-then-authenticate (EtA) approach for symmetric cryptography.

1 Introduction

Where *encryption* converts a message *(plaintext)* into a scrambled message *(ciphertext)* such that revealing the latter does not expose the former, *steganography* goes further and seeks to hide even the fact that secret communication is taking place. A cryptography-based approach is to encrypt the plaintext to be hidden, in this context also known as the *hiddentext,* and embed the resulting ciphertext in a seemingly innocuous message, the *covertext.* The recipient extracts the ciphertext from the covertext and then uses the appropriate cryptographic decryption key to recover the hiddentext. Formal treatments of public-key steganography based on this approach have recently appeared in [3] and [5].

Public-key encryption schemes usually do not output ciphertexts that are pseudo-random in the sense of being indistinguishable from uniformly random bit strings of the same length. Given just the public key, it is typically easy to tell that certain bit strings cannot have come up during proper encryption (e.g., a bit string interpreted as an integer would exceed the modulus that would have been used during encryption). Thus the prospects will often be good for an adversary who tries to distinguish actual ciphertexts from random bit strings: In random bit strings, if an invalid encoding appears, this will clearly reveal that the bit string in question is not an actual ciphertext. Conversely, depending on the

* Supported by a DAAD (German Academic Exchange Service) Postdoc fellowship.

P. Samarati et al. (Eds.): ESORICS 2004, LNCS 3193, pp. 335–351, 2004.

probability that random data will give a valid encoding, seeing a valid encoding or repeatedly seeing valid encodings can provide possibly strong indication that these values are not random.

The present work provides a public-key encryption scheme with pseudo-random ciphertexts as required for public-key steganography constructions like those from [3] and [5]. Some previous schemes do exist (e.g. based on RSA encryption or Diffie-Hellman, see [3, Section 4]), but their downside is the length of ciphertexts; and the expense of having relatively long ciphertexts can be especially high in the context of steganography. (The number of bits available for embedded ciphertexts typically is a small fraction of the covertext length. To avoid detection of the steganographic communication, one would generally want to have to use as little cover channel communication as possible.)

Our scheme uses elliptic curve cryptography to obtain short ciphertexts. Ciphertext pseudo-randomness is not immediate with any single cryptographically secure elliptic curve. Our trick to achieve it is to use a pair of elliptic curves over a field $\mathbb{F}_{2^m}$ such that each curve is a twist of the other. (The idea to use elliptic curves with their twists has also come up in [21] and [22].)

The construction is hybrid and resembles DHIES [2] (also known as DHAES from the earlier version [1] of that publication): it involves a key encapsulation mechanism (KEM), a one-time message authentication code (MAC), and a stream cipher. While our elliptic curve KEM is novel and will be described in detail, a wide range of existing MACs and stream ciphers are available that can be plugged into the construction.

Like DHIES, our scheme achieves security in a strong sense, namely security under adaptive chosen-ciphertext attack (CCA). Besides by using a new KEM, it differs from DHIES in that it applies the symmetric cryptographic primitives in a different order: While DHIES uses an *encrypt-then-authenticate* (EtA) approach by computing the MAC on a symmetric ciphertext, our construction uses an *authenticate-then-encrypt* (AtE) approach by computing the MAC of the plaintext and then symmetrically encrypting the MAC and the plaintext. This modification is needed to prove ciphertext pseudo-randomness without requiring additional security assumptions. Just as for DHIES, the security assumptions are mostly standard with the exception of a less conventional oracle Diffie-Hellman assumption (which finds justification in the random-oracle model based on a more usual gap-Diffie-Hellman assumption [30], but is a concrete computational assumption that can be expressed without resorting to this idealized model).

We also provide an appropriate pair of elliptic curves over $\mathbb{F}_{2^{163}}$ to make the proposed scheme practical, thanks to Reynald Lercier who ran his point counting algorithms to find suitable curves for the requirements of the scheme. The curve pair is verifiably pseudo-random (like the well-known curves as in [13]) to allow for independent verification that it has been generated properly from a seed value using a standardized process.

Section 2 gives formalizations of public-key encryption and key encapsulation with the relevant notions of security, CCA security and ciphertext pseudo-randomness. Section 3 presents our elliptic curve KEM as the essential new

cryptographic primitive. Subsequently, Section 4 looks at well-known symmetric primitives, namely MACs and pseudo-random bit string generators for use as stream ciphers. Section 5 puts the primitives together to build a public-key encryption scheme, giving quantitative security results for CCA security and ciphertext pseudo-randomness. Finally, Section 6 summarizes our conclusions.

2 Public-Key Cryptography with Pseudo-random Ciphertexts: Concepts

2.1 Public-Key Encryption

We formalize public-key encryption with two security notions, security under adaptive chosen-ciphertext attack and ciphertext pseudo-randomness. Section 5 will show how to build a public-key encryption scheme with these properties using appropriate primitives.

We start by describing the notion of public-key encryption in terms of its interface. Note that algorithms are in general probabilistic.

Definition 1. *A* public-key encryption scheme PKE *specifies a* key generation algorithm PKE.KeyGen *and algorithms* PKE.Encrypt *and* PKE.Decrypt. *Algorithm* PKE.KeyGen *takes no input; it outputs a key pair* (PK, SK) *consisting of a* public key PK *and a* secret key SK. *For a* plaintext m *(an arbitrary bit string, possibly subject to length limitations for the specific public-key encryption scheme),*

$$\mathsf{PKE.Encrypt}(PK, m)$$

returns a bit string c *as* ciphertext. *On arbitrary input* c'*,*

$$\mathsf{PKE.Decrypt}(SK, c')$$

may either return some string m' *or fail and return the special value* $\perp$. *If the key pair* (PK, SK) *has been output by* PKE.KeyGen *and* c *has been output by* $\mathsf{PKE.Encrypt}(PK, m)$*, then evaluating* $\mathsf{PKE.Decrypt}(SK, c)$ *will return the original bit string* m.

To capture security notions quantitatively, we assume that an adversary interacts with the cryptographic scheme in question in a specific attack game and define the adversary's *advantage* in this game. An adversary is an interactive probabilistic algorithm with bounded running time. Saying that a cryptographic scheme is *secure* under some security notion means that the advantage will be very low for every practical adversary (i.e. every adversary from some limited class of admissable adversaries), with details left open on what adversaries would be considered practical and what advantage would be considered sufficiently low. (Cryptographic schemes are often described as parameterized by an integer security parameter determining features such as the length of keys. Then security can be formalized as a requirement that any polynomial-time adversary's advantage be negligible in the security parameter, i.e. asymptotically smaller than the reciprocal of any polynomial. We avoid explicit security parameters, which amounts

to having a fixed security parameter built into algorithms such as PKE.KeyGen.) The security result for our public-key encryption scheme in Section 5 will relate the security of the public-key encryption scheme to the security of its underlying primitives: intuitively, if all primitives are secure under their respective security notions, then the complete scheme will be secure.

As the first security notion for public-key encryption, we describe security under *adaptive chosen-ciphertext attack* (CCA), *CCA security* for short. We use a well-known *find-then-guess* attack game that expresses security as *indistinguishability under CCA* (IND-CCA). The term *indistinguishability* refers to the idea that adversaries should not be able to tell apart encryptions of any two plaintexts, a notion due to [19]; the CCA scenario, which provides the adversary with a decryption oracle, is due to [31]. For equivalent formalizations of CCA security, see [8], [33], and [18, Chapter 5].

Definition 2. *In the* IND-CCA attack game, *an adversary interacts with a public-key encryption scheme* PKE *as follows.*

1. *The adversary queries a* key generation oracle, *which uses* PKE.KeyGen *to determine a key pair* (PK, SK) *and responds with* PK *(while secretly storing* SK*).*
2. *The adversary makes a sequence of queries to a* decryption oracle. *Each query is an arbitrary bit string* s*, and the oracle responds with* PKE.Decrypt (SK, s) *before the adversary proceeds.*
3. *The adversary chooses plaintexts* m_0 *and* m_1 *with* $|m_0| = |m_1|$ *(i.e., of the same length) and sends them to an* encryption oracle. *This oracle chooses* $b \in \{0, 1\}$ *uniformly at random and determines*

$$c = \mathsf{PKE.Encrypt}(PK, m_b),$$

   which is returned to the adversary as the challenge ciphertext.
4. *The adversary again makes a sequence of queries to a* decryption oracle *as in stage 2, where this time the decryption oracle refuses being asked for the decryption of the challenge ciphertext* c *(responding* $\perp$ *for this case).*
5. *The adversary outputs a value* $\widetilde{b} \in \{0, 1\}$.

If A *is any adversary in the IND-CCA attack game, its* CCA advantage *against the public key encryption scheme is*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{PKE},A} = \left|\Pr\left[\widetilde{b} = 1 \mid b = 1\right] - \Pr\left[\widetilde{b} = 1 \mid b = 0\right]\right|.$$

The value $\widetilde{b}$ output by the adversary can be thought of as its guess for b.

The second (less usual) security notion for public-key encryption is *ciphertext pseudo-randomness.* We describe it through a *real-or-ideal* (ROI) attack game.

Definition 3. *In the* real-or-ideal attack game *for a public-key encryption scheme* PKE*, an adversary interacts with* PKE *as follows.*

1. *The adversary queries a* key generation oracle, *which uses* PKE.KeyGen *to determine a key pair* (PK, SK) *and responds with* PK*.*

2. *The adversary makes a query to a* real-or-ideal encryption oracle, *or* encryption oracle *for short. The query consists of a plaintext* m *of any length valid for* PKE. *The encryption oracle determines*

$$c_0 = \mathsf{PKE.Encrypt}(PK, m),$$

generates a uniformly random bit string c_1 *with* $|c_1| = |c_0|$, *chooses* $b \in \{0, 1\}$ *uniformly at random, and responds with* c_b.
3. *The adversary outputs a value* $\widetilde{b} \in \{0, 1\}$.

An adversary A *in this attack game is also called a* real-or-ideal distinguisher *or simply a* distinguisher. *Its* real-or-ideal advantage *against* PKE *is*

$$\mathsf{Adv}^{\mathsf{ROI}}_{\mathsf{PKE},A} = \left|\Pr\left[\widetilde{b} = 1 \mid b = 1\right] - \Pr\left[\widetilde{b} = 1 \mid b = 0\right]\right|.$$

The real-or-ideal encryption oracle operates as a real encryption oracle if $b = 0$ and as an ideal encryption oracle if $b = 1$.

2.2 Key Encapsulation

Following Shoup [32], we use the term *key encapsulation mechanism* (KEM) for a scheme in public-key cryptography that is similar to public-key encryption except that plaintexts cannot be arbitrarily specified by the party that creates a ciphertext: instead, the randomized "encryption" algorithm, given some public key, outputs both a pseudo-random plaintext and a corresponding ciphertext such that the plaintext can be recovered from the ciphertext given the appropriate secret key. Such plaintexts can be used as keys for symmetric cryptography; hence the term *key encapsulation.*

Definition 4. *A* key encapsulation mechanism KEM *specifies a* key generation algorithm KEM.KeyGen *and algorithms* KEM.Encrypt *and* KEM.Decrypt. *Algorithm* KEM.KeyGen *takes no input; it outputs a key pair* (PK, SK) *consisting of a public key* PK *and a secret key* SK. *Algorithm*

$$\mathsf{KEM.Encrypt}(PK)$$

generates a bit string K *of a fixed length* KEM.OutLen *and a ciphertext* $\mathfrak{K}$ *of a fixed length* KEM.CipherLen, *and outputs the pair* $(K, \mathfrak{K})$. *Evaluating*

$$\mathsf{KEM.Decrypt}(SK, \mathfrak{K})$$

will return said bit string K *if the key pair* (PK, SK) *has been generated by* KEM.KeyGen. *On arbitrary input* $\mathfrak{K}'$, *the computation* $\mathsf{KEM.Decrypt}(SK, \mathfrak{K}')$ *may either return some bit string* K' *or fail and return the special value* $\perp$.

Similarly to Section 2.1, we use attack games to express two security notions: security under adaptive chosen-ciphertext attack (CCA security) and ciphertext pseudo-randomness. Section 3 will describe a KEM based on elliptic curve Diffie-Hellman designed to meet these notions.

Definition 5. *In the* real-or-random CCA attack game, *an adversary interacts with a key encapsulation mechanism* KEM *as follows (cf. [14, Section 7.1.2]).*

1. *The adversary queries a* key generation oracle, *which uses* KEM.KeyGen *to compute a key pair* (PK, SK) *and responds with* PK.
2. *The adversary makes a sequence of queries to a* decryption oracle. *Each query is an arbitrary bit string* s *of length* KEM.CipherLen*; the oracle responds with* $\mathsf{KEM.Decrypt}(SK, s)$ *before the adversary proceeds.*
3. *The adversary queries a* real-or-random key encapsulation oracle, *or* key encapsulation oracle *for short. This oracle uses* $\mathsf{KEM.Encrypt}(PK)$ *to obtain a pair* $(K_0, \mathfrak{K}_{\mathsf{oracle}})$*, generates a uniformly random bit string* K_1 *with* $|K_1| = |K_0|$*, chooses* $b_{\mathsf{KEM}} \in \{0, 1\}$ *uniformly at random, and responds with*

$$(K_{b_{\mathsf{KEM}}}, \mathfrak{K}_{\mathsf{oracle}})$$

   as challenge.
4. *The adversary again makes a sequence of queries to a* decryption oracle *as in stage 2, where this time the oracle refuses the specific query* $\mathfrak{K}_{\mathsf{oracle}}$ *(responding* $\perp$ *for this case).*
5. *The adversary outputs a value* $\widetilde{b}_{\mathsf{KEM}} \in \{0, 1\}$.

If A *is any adversary in the real-or-random CCA attack game, its* CCA advantage *against the key encapsulation mechanism KEM is*

$$\mathsf{Adv}^{\mathsf{ROR\text{-}CCA}}_{\mathsf{KEM},A} \;=\; \Big|\Pr\big[\widetilde{b}_{\mathsf{KEM}} = 1 \mid b_{\mathsf{KEM}} = 1\big] - \Pr\big[\widetilde{b}_{\mathsf{KEM}} = 1 \mid b_{\mathsf{KEM}} = 0\big]\Big|.$$

The real-or-random key encapsulation oracle operates as a real key encapsulation oracle if $b_{\mathsf{KEM}} = 0$ and as a random key encapsulation oracle if $b_{\mathsf{KEM}} = 1$.

Definition 6. *In the* real-or-ideal attack game *for a key encapsulation mechanism* KEM*, an adversary interacts with* KEM *as follows.*

1. *The adversary queries a* key generation oracle, *which uses* KEM.KeyGen *to determine a key pair* (PK, SK) *and responds with* PK.
2. *The adversary queries a* real-or-ideal key encapsulation oracle, *or* key encapsulation oracle *for short. The key encapsulation oracle uses* $\mathsf{KEM.Encrypt}(PK)$ *to determine a pair* $(K, \mathfrak{K}_0)$*, generates a uniformly random bit string* $\mathfrak{K}_1$ *of length* KEM.CipherLen*, chooses* $b_{\mathsf{KEM}} \in \{0, 1\}$ *uniformly at random, and responds with* $\mathfrak{K}_{b_{\mathsf{KEM}}}$.
3. *The adversary outputs a value* $\widetilde{b}_{\mathsf{KEM}} \in \{0, 1\}$.

An adversary A *in this attack game is also called a* real-or-ideal distinguisher *or simply a* distinguisher. *Its* real-or-ideal advantage *against* KEM *is*

$$\mathsf{Adv}^{\mathsf{ROI}}_{\mathsf{KEM},A} \;=\; \Big|\Pr\big[\widetilde{b}_{\mathsf{KEM}} = 1 \mid b_{\mathsf{KEM}} = 1\big] - \Pr\big[\widetilde{b}_{\mathsf{KEM}} = 1 \mid b_{\mathsf{KEM}} = 0\big]\Big|.$$

The real-or-ideal key encapsulation oracle operates as a real key encapsulation oracle if $b_{\mathsf{KEM}} = 0$ and as an ideal key encapsulation oracle if $b_{\mathsf{KEM}} = 1$.

3 An Elliptic Curve KEM with Random Ciphertexts

We need a key encapsulation mechanism (KEM) that is CCA-secure and provides ciphertext pseudo-randomness.

First consider CCA security (Definition 5). The scheme DHIES from [2] uses *hash Diffie-Hellman* (HDH) as a CCA-secure KEM. The idea of HDH is to employ Diffie-Hellman [15] in some group followed by the application of a *key derivation function* (KDF) to obtain pseudo-random bit strings from group elements. We will use HDH with *elliptic curve Diffie-Hellman* (ECDH, due to [28] and [23]) following the idea originally proposed in [28] that it suffices to transfer just x-coordinates of points. The assumption that HDH schemes are CCA-secure key encapsulation mechanisms amounts to accepting an *oracle Diffie-Hellman assumption;* see [2]. (By appealing to the *random-oracle model* [9], this assumption can be justified based on a gap-Diffie-Hellman assumption [30], i.e. an assumption on the hardness of the computational Diffie-Hellman problem given a decisional Diffie-Hellman oracle; cf. [2] and [14, Theorem 9]. There are concerns about the use of the random-oracle model [11], but here this idealized model would be only a locally used tool to explain the plausibility of a specific security assumption on the KEM; the oracle Diffie-Hellman assumption can be expressed directly without employing the random-oracle model.)

Now consider ciphertext pseudo-randomness (Definition 6). Our KEM will be constructed to have ciphertexts that are uniformly random bit strings of length $\mathsf{KEM.CipherLen}$, which implies $\mathsf{Adv}^{\mathsf{ROI}}_{\mathsf{KEM},A} = 0$ for any adversary.

We arrive at the KEM by first presenting some preliminaries on elliptic curves (Section 3.1), then discussing system parameters (Section 3.2 with specific values in Appendix A), and finally specifying the actual KEM (Section 3.3).

3.1 Preliminaries on Elliptic Curves

The KEM will use elliptic curves over some finite field $\mathbb{F}_{2^m}$ (refer to e.g. [10] for introductory material on elliptic curve cryptography). There are well-known requirements for cryptographically secure elliptic curves over such fields [20], which we will take into account (Section 3.2). We require that m be an odd prime as there are security concerns about $\mathbb{F}_{2^m}$ with m composite. Every non-supersingular curve over $\mathbb{F}_{2^m}$ is isomorphic to a curve described by a curve equation

$$E_{a,b}: \quad y^2 + xy = x^3 + ax^2 + b$$

where coefficients a and b are elements of $\mathbb{F}_{2^m}$, $b \neq 0$. The group $E_{a,b}(\mathbb{F}_{2^m})$ of rational points on such a curve consists of the set of solutions (x, y) of that equation with $x, y \in \mathbb{F}_{2^m}$, and an additional point $\mathcal{O}$. We have

$$\left|\#E_{a,b}(\mathbb{F}_{2^m}) - 2^m - 1\right| < 2 \cdot 2^{m/2}$$

(Hasse's inequality). The group operation (see [10] or [20] for the definition) is commutative, and by convention is written as addition. $\mathcal{O}$, the *point at infinity,* is the neutral element; the inverse of any element (x, y) is $(x, x + y)$. The group

operation canonically gives rise to scalar multiplication kP of a point P by an integer k.

If $\mathrm{Tr}_{\mathbb{F}_{2^m}/\mathbb{F}_2} a = \mathrm{Tr}_{\mathbb{F}_{2^m}/\mathbb{F}_2} a'$, the curves

$$\begin{aligned} E_{a,b}\colon &\quad y^2 + xy = x^3 + ax^2 + b, \\ E_{a',b}\colon &\quad y^2 + xy = x^3 + a'x^2 + b \end{aligned}$$

are isomorphic with a group isomorphism $E_{a,b}(\mathbb{F}_{2^m}) \to E_{a',b}(\mathbb{F}_{2^m})$ given by

$$(x, y) \mapsto (x, y + sx)$$

where $s \in \mathbb{F}_{2^m}$ satisfies $a' = a + s + s^2$ (such an s always exists in this case); otherwise $E_{a,b}$ and $E_{a',b}$ are called *twists* of each other.

If $E_{a,b}$ and $E_{a',b}$ are twists of each other, then for every $x \in \mathbb{F}_{2^m}$ there is a $y \in \mathbb{F}_{2^m}$ such that (x, y) is a point on one of the two curves. Specifically, each x occurs exactly twice: either there are two different points (x, y) and $(x, x+y)$ on the same curve; or $(x, y) = (0, \sqrt{b})$, which is a point on both curves. The total number of points (taking into account $\mathcal{O}$ for each curve) is

$$\#E_{a,b}(\mathbb{F}_{2^m}) + \#E_{a',b}(\mathbb{F}_{2^m}) \;=\; 2^{m+1} + 2.$$

Hasse's inequality implies $\#E_{a,b}(\mathbb{F}_{2^m}) \approx 2^m \approx \#E_{a',b}(\mathbb{F}_{2^m})$.

3.2 System Parameters

If in the situation of Section 3.1 we vary coefficient a for fixed b, we obtain curves in one of two equivalence classes depending on whether $\mathrm{Tr}_{\mathbb{F}_{2^m}/\mathbb{F}_2} a = 0$ or $\mathrm{Tr}_{\mathbb{F}_{2^m}/\mathbb{F}_2} a = 1$. All curves within a single class are isomorphic; the other class contains their twists. We have $\mathrm{Tr}_{\mathbb{F}_{2^m}/\mathbb{F}_2} 0 = 0$ and $\mathrm{Tr}_{\mathbb{F}_{2^m}/\mathbb{F}_2} 1 = 1$ as m is odd, so we can use $E_{0,b}$ and $E_{1,b}$ as a canonical pair of curves that are twists of each other. (The explicit isomorphism shown in Section 3.1 means that the specific choices for a affect only y-coordinates.) Coefficient b remains to be chosen such that the groups $E_{0,b}(\mathbb{F}_{2^m})$ and $E_{1,b}(\mathbb{F}_{2^m})$ are both cryptographically secure. Requirements for suitable curves are ([12, Section 3.1.2.1], [4, Annex A.3.2]):

- The group order $\#E_{a,b}(\mathbb{F}_{2^m})$ must be a product $h_{a,b}\, p_{a,b}$ with $1 \le h_{a,b} \le 4$ and $p_{a,b}$ prime.
- For said prime, it must hold that $2^{mB} \not\equiv 1 \pmod{p_{a,b}}$ for $1 \le B \le 20$ (the *MOV condition* [27]).

The curve $E_{0,b}$ has a point $\left(\sqrt[4]{b}, \sqrt{b}\right)$ of order 4, so we will use $h_{0,b} = 4$; note that the group $E_{0,b}(\mathbb{F}_{2^m})$ will then necessarily be cyclic. From $\#E_{0,b}(\mathbb{F}_{2^m}) + \#E_{1,b}(\mathbb{F}_{2^m}) \equiv 2 \pmod 4$, it follows that $h_{1,b} = 2$; thus $E_{1,b}(\mathbb{F}_{2^m})$ will be cyclic too. Define t_b such that

$$\begin{aligned} \#E_{0,b}(\mathbb{F}_{2^m}) &= 2^m + 1 - t_b = 4\, p_{0,b}, \\ \#E_{1,b}(\mathbb{F}_{2^m}) &= 2^m + 1 + t_b = 2\, p_{1,b}; \end{aligned}$$

a pair of suitable curves can be generated by choosing $b \in \mathbb{F}_{2^m} \setminus \{0\}$, determining t_b through *point counting* techniques (see [26] for fast algorithms), and verifying that

$$p_{0,b} = \frac{2^m + 1 - t_b}{4} \qquad \text{and} \qquad p_{1,b} = \frac{2^m + 1 + t_b}{2}$$

are indeed both prime and satisfy the MOV condition. Heuristically, for random b, the integer $p_{0,b}$ will be prime with probability about $1/m$ ([17] gives more precise estimates), and $p_{1,b}$ will be prime as well with probability about $1/m$; the MOV condition is not likely to cause problems if b is actually random. Thus, one has to expect to have to test approximately m^2 random choices for b before finding a suitable one.

It is common to use standardized pre-generated elliptic curves over appropriate fields instead of generating new curves as part of key generation. Parameters provided to others are usually expected to be *verifiably pseudo-random* with b derived from some *seed* value through a standardized process, typically with SHA-1 as specified in [4, Annex A.3.3.1] and [20, Annex A.12.6]. We provide a verifiably pseudo-random pair of curves over $\mathbb{F}_{2^{163}}$ in Appendix A.

Now let $b \in \mathbb{F}_{2^m} \setminus \{0\}$ be fixed such that the curves $E_0 = E_{0,b}$ and $E_1 = E_{1,b}$ are both cryptographically suitable with orders

$$\#E_0(\mathbb{F}_{2^m}) = 4\,p_0 = n_0 \qquad \text{and} \qquad \#E_1(\mathbb{F}_{2^m}) = 2\,p_1 = n_1$$

where p_0 and p_1 are prime. As additional parameters, for $a \in \{0,1\}$, let G_a be a generator of $E_a(\mathbb{F}_{2^m})$, i.e. any element of order n_a. Note that $n_0 + n_1 = 2^{m+1} + 2$.

3.3 Specification

We assume that system parameters as discussed in Section 3.2 have been fixed, and that mappings

$$\mathsf{encode}\colon \mathbb{F}_{2^m} \to \{0,1\}^m \qquad \text{and} \qquad \mathsf{decode}\colon \{0,1\}^m \to \mathbb{F}_{2^m}$$

that are inverses of each other have been agreed upon (cf. FE2OSP and OS2FEP in [20] or FieldElement-to-OctetString and OctetString-to-FieldElement in [12] and [4]). We also assume that a key derivation function (KDF) that outputs bit strings of some length $\mathsf{KEM.OutLen}$ has been specified (for a practical example, see KDF1 in [20] or ANSI-X9.63-KDF in [12]). The KDF will be used only on input bit strings of length $2m$. We now show a key encapsulation mechanism KEM with $\mathsf{KEM.CipherLen} = m$.

KEM.KeyGen: Choose integers s_0 and s_1 independently uniformly at random among those satisfying $0 < s_a < n_a$ with s_a and n_a relatively prime. Then compute the points $P_a = s_a G_a$ for $a \in \{0,1\}$. Output the *public key* $PK = (P_0, P_1)$ and the *secret key* $SK = (s_0, s_1)$.

KEM.Encrypt(PK): Choose $a \in \{0,1\}$ with probability $\frac{n_0 - 1}{2^{m+1}}$ for $a = 0$, $\frac{n_1 - 1}{2^{m+1}}$ for $a = 1$. Then choose a uniformly random integer u with $0 < u < n_a$. Compute $Q = uG_a$ and $R = uP_a$ in the group $E_a(\mathbb{F}_{2^m})$; these points will never be $\mathcal{O}$ with valid system parameters and a valid public key, so they can be written as coordinate pairs (x_Q, y_Q) and (x_R, y_R). Finally, set

$$K = \mathrm{KDF}(\mathfrak{K} \parallel \mathsf{encode}(x_R)\big),$$
$$\mathfrak{K} = \mathsf{encode}(x_Q)$$

($\parallel$ denotes concatenation) and return the pair $(K, \mathfrak{K})$.

KEM.Decrypt($SK, \mathfrak{K}$): Set $x = \mathsf{decode}(\mathfrak{K})$. Then determine a $y \in \mathbb{F}_{2^m}$ such that (x, y) is a point on E_0 if there is such a y; if so, set $a = 0$. Otherwise, determine a $y \in \mathbb{F}_{2^m}$ such that (x, y) is a point on E_1 and set $a = 1$.In either case, set $Q = (x, y)$ and compute $R = s_a Q$ in the group $E_a(\mathbb{F}_{2^m})$. This point will never be $\mathcal{O}$ with valid system parameters and a valid secret key, so it can be written as a coordinate pair (x_R, y_R). Finally, set

$$K = \mathrm{KDF}(\mathfrak{K} \parallel \mathsf{encode}(x_R)\big)$$

and return K.

Determining y given x in KEM.Decrypt amounts to performing *point decompression;* for algorithms, cf. [4, Section 4.22], [20, Annex A.12.9], or [12, Section 2.3.4].

In KEM.Encrypt, each element of $\mathbb{F}_{2^m}$ can come up as x_Q with probability 2^{-m} (any given x_Q appears exactly twice among the x-coordinates of the $n_0 - 1$ points of $E_{0,b}(\mathbb{F}_{2^m}) \setminus \{\mathcal{O}\}$ and the $n_1 - 1$ points of $E_{1,b}(\mathbb{F}_{2^m}) \setminus \{\mathcal{O}\}$). Thus, the distribution of $\mathfrak{K} \in \{0,1\}^m$ is uniform.

Assuming that a has been correctly recovered, the point Q computed in KEM.Decrypt will be either identical to or the inverse of the point Q originally used in KEM.Encrypt; the same relationship will apply to the respective points R (we have $\pm R = \pm uP_a = \pm us_a G_a = \pm s_a Q$); so x_R will come out correctly as inversion affects only y-coordinates. KEM.Decrypt can unequivocally recover a in all cases except one, when $x = 0$ so that $Q = (0, \sqrt{b})$ is the point of order 2 on either $E_0(\mathbb{F}_{2^m})$ or $E_1(\mathbb{F}_{2^m})$; but in this case (of negligible probability) the same result would be obtained in either group.

As discussed earlier, it is reasonable to make an oracle Diffie-Hellman assumption implying that this KEM provides CCA security. Ciphertexts generated as described above are uniformly random. By Hasse's inequality (Section 3.1), a simplified variant of KEM.Encrypt that picks a uniformly random $a \in \{0,1\}$ would still achieve ciphertext pseudo-randomness.

4 Symmetric Primitives

4.1 Message Authentication Code

The usual notion of a message authentication code (MAC) allows using a single key for authenticating multiple messages. Like DHIES, we only need a one-time MAC.

Definition 7. *A* one-time message authentication code MAC *specifies a key length* MAC.KeyLen*, an output length* MAC.OutLen*, and a deterministic algorithm that, given a bit string K of length* MAC.KeyLen *(a* key*) and a bit string m, returns a bit string* $\mathsf{MAC}(K, m)$ *of length* MAC.OutLen *(a* tag*).*

The security of a one-time MAC is expressed as follows.

Definition 8. *In the* forgery attack game*, an adversary interacts with a one-time MAC as follows.*

1. *The adversary submits a bit string m to a* MAC oracle*. This oracle generates a uniformly random bit string K of length* MAC.KeyLen *and responds with* $\mathsf{MAC}(K, m)$*.*
2. *The adversary outputs a list $(m_1, t_1), (m_2, t_2), \ldots, (m_l, t_l)$ of pairs of bit strings.*

Let A be any adversary in the forgery attack game. (Its running time bound implies a bound on the length l of the list.) We say that adversary A has produced a forgery *if there is some k such that* $\mathsf{MAC}(K, m_k) = t_k$ *and* $m_k \neq m$*. The adversary's* forgery advantage *against* MAC*, denoted* $\mathsf{Adv}^{\mathsf{Forge}}_{\mathsf{MAC},A}$*, is the probability that it produces a forgery in the above game.*

A popular MAC construction is HMAC [6]; a specific variant with MAC.KeyLen = 160 and MAC.OutLen = 80 is HMAC-SHA1-80 [24].

4.2 Pseudo-random Bit String Generator

Our hybrid construction for public-key encryption uses a stream cipher to perform symmetric encryption. While other notions of stream ciphers are conceivable, for simplicity we assume a stream cipher based on the usual XOR paradigm: symmetric encryption and decryption are the same operation, namely XOR with a pseudo-random bit string generated from a key.

Definition 9. *A* pseudo-random bit string generator STREAM *specifies a key length* STREAM.KeyLen *and a deterministic algorithm that, given a bit string K of length* STREAM.KeyLen *(a* key*) and an integer n, generates an output bit string* $\mathsf{STREAM}(K, n)$ *of length n.*

Security is described through a *real-or-ideal* attack game.

Definition 10. *In the* real-or-ideal attack game *for a pseudo-random bit string generator* STREAM*, an adversary interacts with* STREAM *as follows.*

1. *The adversary sends an integer n to a* real-or-ideal bit string oracle*. The oracle generates a uniformly random K with $|K| =$* STREAM.KeyLen*, sets*

$$stream_0 = \mathsf{STREAM}(K, n),$$

generates a uniformly random $stream_1$ with $|stream_1| = n$, chooses $b_{\mathsf{STR}} \in \{0, 1\}$ uniformly at random, and responds with $stream_b$ as challenge*. (It is understood that the adversary's running time bound implies a bound on n.)*

2. *The adversary outputs a value* $\widetilde{b}_{\mathsf{STR}} \in \{0,1\}$.

An adversary A in this attack game is also called a real-or-ideal distinguisher *or simply a* distinguisher. *Its* real-or-ideal advantage *against* STREAM *is*

$$\mathsf{Adv}^{\mathsf{ROI}}_{\mathsf{STREAM},A} = \Big|\Pr\big[\widetilde{b}_{\mathsf{STR}} = 1 \mid b_{\mathsf{STR}} = 1\big] - \Pr\big[\widetilde{b}_{\mathsf{STR}} = 1 \mid b_{\mathsf{STR}} = 0\big]\Big|.$$

An example implementation is the *counter mode* (CTR) of a symmetric block cipher such as AES (see [7] and [29]). (For $n \leq \mathsf{STREAM.KeyLen}$, it is also be possible to define $\mathsf{STREAM}(K, n)$ as simply the n-bit prefix of K; then any distinguisher would have real-or-ideal advantage 0.)

5 Public-Key Encryption with Pseudo-random Ciphertexts: Hybrid Construction

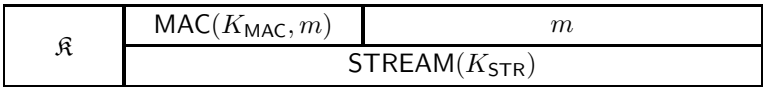
Now we are ready to show how a public-key encryption scheme PKE as discussed in Section 2.1 can be built from primitives KEM (Section 2.2 and Section 3), MAC (Section 4.1), and STREAM (Section 4.2). Section 5.1 presents the hybrid construction, which follows DHIES except that it uses an *authenticate-then-encrypt* (AtE) rather than an *encrypt-then-authenticate* (EtA) approach. Section 5.2 gives security results for CCA security and ciphertext pseudo-randomness.

5.1 Specification

We require that $\mathsf{KEM.OutLen} = \mathsf{MAC.KeyLen} + \mathsf{STREAM.KeyLen}$. The *key generation algorithm* PKE.KeyGen is the same as KEM.KeyGen. The *encryption algorithm* determines $\mathsf{PKE.Encrypt}(PK, m)$ as follows.

1. Use $\mathsf{KEM.Encrypt}(PK)$ to generate a pair $(K, \mathfrak{K})$.
2. Split K into bit strings K_{MAC} of length MAC.KeyLen and K_{STR} of length STREAM.KeyLen; i.e., $K = K_{\mathsf{MAC}} \parallel K_{\mathsf{STR}}$.
3. Compute $\mathfrak{M} = \mathsf{MAC}(K_{\mathsf{MAC}}, m)$.
4. Compute $\mathfrak{C} = (\mathfrak{M} \parallel m) \oplus \mathsf{STREAM}(K_{\mathsf{STR}}, \mathsf{MAC.OutLen} + |m|)$.
5. Return the ciphertext $\mathfrak{K} \parallel \mathfrak{C}$.

We depict the resulting ciphertext structure with concatenation horizontally and XOR vertically:



The *decryption algorithm* computes $\mathsf{PKE.Decrypt}(PK, c)$ as follows.

1. Abort with an error (return $\perp$) if $|c| < \mathsf{KEM.CipherLen} + \mathsf{MAC.OutLen}$.
2. Split c into a part $\mathfrak{K}$ of length KEM.CipherLen and a part $\mathfrak{C}$ (i.e., $c = \mathfrak{K} \parallel \mathfrak{C}$).

3. Compute
$$K = \mathsf{KEM.Decrypt}(SK, \mathfrak{K}).$$
If this computation fails, abort with an error (return $\perp$).
4. Split K into bit strings K_{MAC} of length $\mathsf{MAC.KeyLen}$ and K_{STR} of length $\mathsf{STREAM.KeyLen}$ (i.e., $K = K_{\mathsf{MAC}} \parallel K_{\mathsf{STR}}$).
5. Compute
$$P = \mathfrak{C} \oplus \mathsf{STREAM}(K_{\mathsf{STR}}, |\mathfrak{C}|).$$
6. Split P into a part $\mathfrak{M}$ of length $\mathsf{MAC.OutLen}$ and a part m (i.e., $P = \mathfrak{M} \parallel m$).
7. Compute
$$\widetilde{\mathfrak{M}} = \mathsf{MAC}(K_{\mathsf{MAC}}, m).$$
If $\widetilde{\mathfrak{M}} \neq \mathfrak{M}$, abort with an error (return $\perp$).
8. Return m as decryption result.

Let c be a ciphertext generated as $\mathsf{PKE.Encrypt}(PK, m)$. It is straightforward to verify that $\mathsf{PKE.Decrypt}(SK, c)$ will indeed recover m if KEM is a key encapsulation mechanism according to Definition 4 and the key pair (PK, SK) has been generated properly. (Note that decryption step 3 cannot actually fail for the KEM from Section 3 with valid system parameters and a valid secret key.)

In practical use for steganography, the exact length of the ciphertext to be considered may not always be known in advance when some postfix has been added. In this case, multiple conceivable lengths can be tried during decryption. Observe that many such decryption attempts can easily be combined into a single algorithm such that $\mathsf{KEM.Decrypt}$ is used only once.

5.2 Security Results

We relate the security of the public-key encryption scheme PKE to the security of the underlying primitives KEM, MAC, and STREAM.

First consider CCA security. Let A be an adversary attacking PKE in the IND-CCA attack game (Definition 2). It can be shown that there are adversaries A_1 against KEM in a real-or-random CCA attack game, A_2 against MAC, and A_3 against STREAM, all having essentially the same running time as A, such that
$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{PKE},A} \leq 2 \cdot \big(\mathsf{Adv}^{\mathsf{ROR\text{-}CCA}}_{\mathsf{KEM},A_1} + \mathsf{Adv}^{\mathsf{Forge}}_{\mathsf{MAC},A_2} + \mathsf{Adv}^{\mathsf{ROI}}_{\mathsf{STREAM},A_3}\big).$$
The proof uses standard techniques (see e.g. [14]) and requires essentially no changes for the hybrid AtE construction with a stream cipher compared with the conventional hybrid EtA construction. We omit the details for lack of space.

Now consider ciphertext pseudo-randomness. Let A be an adversary attacking PKE in the real-or-ideal attack game (Definition 3). It can be shown that there are adversaries A_1 against KEM in a real-or-random CCA attack game, A_2 against KEM in a real-or-ideal attack game, and A_3 against STREAM, all having essentially the same running time as A, such that
$$\mathsf{Adv}^{\mathsf{ROI}}_{\mathsf{PKE},A} \leq 2 \cdot \big(\mathsf{Adv}^{\mathsf{ROR\text{-}CCA}}_{\mathsf{KEM},A_1} + \mathsf{Adv}^{\mathsf{ROI}}_{\mathsf{KEM},A_2} + \mathsf{Adv}^{\mathsf{ROI}}_{\mathsf{STREAM},A_3}\big).$$
Details of the proof are given in Appendix B.

6 Conclusions

A new variant of elliptic curve Diffie-Hellman employing a pair of curves where each curve is a twist of the other provides a key encapsulation mechanism (KEM) with short random ciphertexts.

Such a KEM can be used for CCA-secure public-key encryption with pseudo-random ciphertexts, as needed for steganography. Our hybrid construction resembles DHIES, but uses an AtE rather than EtA approach in the interest of provable ciphertext pseudo-randomness. In practice, the ciphertext length can be as short as the length of the plaintext plus 243 bits (163 bits for the KEM with elliptic curves over $\mathbb{F}_{2^{163}}$, 80 bits for the MAC).

References

1. Abdalla, M., Bellare, M., and Rogaway, P. DHAES: An encryption scheme based on the Diffie-Hellman problem. Submission to IEEE P1363a. `http://grouper.ieee.org/groups/1363/P1363a/Encryption.html`, 1998.
2. Abdalla, M., Bellare, M., and Rogaway, P. The oracle Diffie-Hellman assumptions and an analysis of DHIES. In *Progress in Cryptology – CT-RSA 2001* (2001), D. Naccache, Ed., vol. 2020 of *LNCS*, pp. 143–158.
3. Ahn, L. v., and Hopper, N. Public key steganography. In *Advances in Cryptology – EUROCRYPT 2004* (2004), C. Cachin and J. Camenisch, Eds., vol. 3027 of *LNCS*, pp. 323–341.
4. American National Standards Institute (ANSI). Public key cryptography for the financial services industry: The elliptic curve digital signature algorithm (ECDSA). ANSI X9.62, 1998.
5. Backes, M., and Cachin, C. Public-key steganography with active attacks. Cryptology ePrint Archive Report 2003/231 (revised 16 Feb 2004), 2004. Available from `http://eprint.iacr.org/`.
6. Bellare, M., Canetti, R., and Krawczyk, H. Keying hash functions for message authentication. In *Advances in Cryptology – CRYPTO '96* (1996), N. Koblitz, Ed., vol. 1109 of *LNCS*, pp. 1–15.
7. Bellare, M., Desai, A., Jokipii, E., and Rogaway, P. A concrete security treatment of symmetric encryption. In *38th Annual Symposium on Foundations of Computer Science (FOCS '97)* (1997), IEEE Computer Society, pp. 394–403.
8. Bellare, M., Desai, A., Pointcheval, D., and Rogaway, P. Relations among notions of security for public-key encryption schemes. In *Advances in Cryptology – CRYPTO '98* (1998), H. Krawczyk, Ed., vol. 1462 of *LNCS*, pp. 26–46.
9. Bellare, M., and Rogaway, P. Random oracles are practical: A paradigm for designing efficient protocols. In *First Annual Conference on Computer and Communications Security* (1993), ACM, pp. 62–73.
10. Blake, I. F., Seroussi, G., and Smart, N. P. *Elliptic Curves in Cryptography*, vol. 265 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, 1999.
11. Canetti, R., Goldreich, O., and Halevi, S. The random oracle methodology, revisited. E-print cs.CR/0010019, 2000. Available from `http://arXiv.org/abs/cs/0010019`.
12. Certicom Research. Standards for efficient cryptography – SEC 1: Elliptic curve cryptography. Version 1.0, 2000. Available from `http://www.secg.org/`.

13. Certicom Research. Standards for efficient cryptography – SEC 2: Recommended elliptic curve cryptography domain parameters. Version 1.0, 2000. Available from `http://www.secg.org/`.
14. Cramer, R., and Shoup, V. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*. To appear. Available from `http://shoup.net/papers/` (2003).
15. Diffie, W., and Hellman, M. E. New directions in cryptography. *IEEE Transactions on Information Theory 22*, 6 (1976), 644–654.
16. Fouquet, M., Gaudry, P., and Harley, R. Finding secure curves with the Satoh-FGH algorithm and an early-abort strategy. In *Advances in Cryptology – EUROCRYPT 2001* (2001), B. Pfitzmann, Ed., vol. 2045 of *LNCS*, pp. 14–29.
17. Galbraith, S., and McKee, J. The probability that the number of points on an elliptic curve over a finite field is prime. CACR Technical Report CORR 99-51, 1999. Available from `http://www.cacr.math.uwaterloo.ca/techreports/1999/`.
18. Goldreich, O. *Foundations of Cryptography – Vol. II: Basic Applications*. Cambridge University Press, 2004.
19. Goldwasser, S., and Micali, S. Probabilistic encryption. *Journal of Computer and System Sciences 28* (1984), 270–299.
20. Institute of Electrical and Electronics Engineers (IEEE). IEEE standard specifications for public-key cryptography. IEEE Std 1363-2000, 2000.
21. Kaliski, Jr., B. S. A pseudo-random bit generator based on elliptic logarithms. In *Advances in Cryptology – CRYPTO '86* (1987), A. M. Odlyzko, Ed., vol. 263 of *LNCS*, pp. 84–103.
22. Kaliski, Jr., B. S. One-way permutations on elliptic curves. *Journal of Cryptology 3* (1991), 187–199.
23. Koblitz, N. Elliptic curve cryptosystems. *Mathematics of Computation 48* (1987), 203–209.
24. Krawczyk, H., Bellare, M., and Canetti, R. HMAC: Keyed-hashing for message authentication. RFC 2104. Available from `http://www.ietf.org/rfc/rfc2104.txt`, 1997.
25. Lercier, R. Finding good random elliptic curves for cryptosystems defined over $\mathbb{F}_{2^n}$. In *Advances in Cryptology – EUROCRYPT '97* (1997), W. Fumy, Ed., vol. 1233 of *LNCS*, pp. 379–392.
26. Lercier, R., and Lubicz, D. Counting points on elliptic curves over finite fields of small characteristic in quasi quadratic time. In *Advances in Cryptology – EUROCRYPT 2003* (2003), E. Biham, Ed., vol. 2656 of *LNCS*, pp. 360–373.
27. Menezes, A., Okamoto, T., and Vanstone, S. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Transactions on Information Theory 39* (1993), 1639–1646.
28. Miller, V. S. Use of elliptic curves in cryptography. In *Advances in Cryptology – CRYPTO '85* (1986), H. C. Williams, Ed., vol. 218 of *LNCS*, pp. 417–428.
29. National Institute of Standards and Technology. Recommendation for block cipher modes of operation – methods and techniques. NIST Special Publication SP 800-38A, 2001.
30. Okamoto, T., and Pointcheval, D. A new class of problems for the security of cryptographic schemes. In *Public Key Cryptography – PKC 2001* (2001), K. Kim, Ed., vol. 1992 of *LNCS*, pp. 104–118.
31. Rackoff, C. W., and Simon, D. R. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Advances in Cryptology – CRYPTO '91* (1992), J. Feigenbaum, Ed., vol. 576 of *LNCS*, pp. 433–444.

32. Shoup, V. A proposal for an ISO standard for public key encryption. Version 2.1, December 20, 2001. http://shoup.net/papers/.
33. Watanabe, Y., Shikata, J., and Imai, H. Equivalence between semantic security and indistinguishability against chosen ciphertext attacks. In *Public Key Cryptography – PKC 2003* (2003), Y. G. Desmedt, Ed., vol. 2567 of *LNCS*, pp. 71–84.

A Example Parameters

We use hexadecimal representations of octet strings and encode field elements of $\mathbb{F}_{2^{163}}$ as polynomials over $\mathbb{F}_2$ based on the reducing polynomial given as

$$\texttt{0800000000000000000000000000000000000000C9};$$

see e.g. [12] for a detailed explanation. A suitable pair of elliptic curves for the KEM in Section 3 is given by

$$b = \texttt{05846d0fda255361606711bf7a99b0722e2ec8f76b}.$$

This b has been generated verifiably pseudo-randomly with SHA-1 from the seed

$$\texttt{f391f2426f9ca3af80bc4537dd7224d43c1639aa}.$$

following [4, Annex A.3.3.1] and [20, Annex A.12.6]. The curves $E_{0,b}$ and $E_{1,b}$ have

$$n_0 = 4\,p_0 = 4 \cdot 2923003274661805836407371179614143033958162426611,$$
$$n_1 = 2\,p_1 = 2 \cdot 5846006549323611672814736302501978089331135490587$$

rational points, respectively, with both p_0 and p_1 prime. The curves $E_{a,b}(\mathbb{F}_{2^{163}})$ have points $G_a = (x_a, y_a)$ of order n_a for example for $x_0 = \texttt{0000...01}$ and $x_1 = \texttt{0000...02}$.

The above coefficient b was provided by Reynald Lercier, who applied the point counting algorithms described in [25] and [16] to a list of verifiably pseudo-random b values and factored the curve orders to detect suitable curve pairs. Out of the 30 000 ($\approx 163^2$) choices for b examined, five had the desired properties.

B Security Proof: Ciphertext Pseudo-randomness

Let $\mathbf{G}_0$ denote the real-or-ideal attack game from Definition 3 where a distinguisher A attacks PKE. The encryption oracle stage in $\mathbf{G}_0$ can be expressed as follows for PKE from Section 5.1: A submits m; the encryption oracle uses $\mathsf{KEM.Encrypt}(PK)$ to generate $(K_{\mathsf{oracle}}, \mathfrak{K}_{\mathsf{oracle}})$, splits $K_{\mathsf{oracle}} = K_{\mathsf{MAC}} \| K_{\mathsf{STR}}$, computes $\mathfrak{M} = \mathsf{MAC}(K_{\mathsf{MAC}}, m)$ and $\mathit{stream} = \mathsf{STREAM}(K_{\mathsf{STR}}, \mathsf{MAC.OutLen} + |m|)$, sets $c_0 = \mathfrak{K}_{\mathsf{oracle}} \parallel \big((\mathfrak{M} \parallel m) \oplus \mathit{stream}\big)$, generates a uniformly random c_1 with $|c_1| = |c_0|$ and a uniformly random $b \in \{0, 1\}$, and finally responds with c_b.

Let $\mathbf{G}_1$ be like $\mathbf{G}_0$ but with K_{oracle} uniformly random (of appropriate length); $\mathbf{G}_2$ like $\mathbf{G}_1$ but with $\mathfrak{K}_{\mathsf{oracle}}$ uniformly random; $\mathbf{G}_3$ like $\mathbf{G}_2$ but with *stream* uniformly random. We can expose A to these different games and look at $\Pr_{\mathbf{G}_i}\left[\widetilde{b} = b\right]$. We will build adversaries A_1 against KEM in a real-or-random CCA attack game, A_2 against KEM in a real-or-ideal attack game, and A_3 against STREAM. These adversaries run A and provide it with an encryption oracle by performing the encryption oracle stage as above, using pre-generated values K_{oracle}, $\mathfrak{K}_{\mathsf{oracle}}$, and b.

A_1 attacks KEM in a real-or-random CCA attack game (Definition 5). First it picks a uniform random $b \in \{0,1\}$ and queries its real-or-random key encapsulation oracle to obtain a pair $(K_{\mathsf{oracle}}, \mathfrak{K}_{\mathsf{oracle}})$. Then it runs A (relaying PK from its key generation oracle), playing the role of the encryption oracle. Finally, when A outputs $\widetilde{b}$, A_1 outputs 1 if $\widetilde{b} = b$ and 0 otherwise. Observe that

$$\left|\Pr_{\mathbf{G}_1}\left[\widetilde{b} = b\right] - \Pr_{\mathbf{G}_0}\left[\widetilde{b} = b\right]\right| = \mathsf{Adv}^{\mathsf{ROR\text{-}CCA}}_{\mathsf{KEM},A_1}$$

($\mathbf{G}_1$ corresponds to $b_{\mathsf{KEM}} = 1$, $\mathbf{G}_0$ to $b_{\mathsf{KEM}} = 0$).

A_2 attacks KEM in a real-or-ideal attack game (Definition 6). First it generates $b \in \{0,1\}$ and a bit string K_{oracle} of length $\mathsf{KEM.OutLen}$ uniformly at random and queries its real-or-ideal key encapsulation oracle to obtain a bit string $\mathfrak{K}_{\mathsf{oracle}}$. Then it runs A (relaying PK from its key generation oracle), playing the role of the encryption oracle. Finally, when A outputs $\widetilde{b}$, A_2 outputs 1 if $\widetilde{b} = b$ and 0 otherwise. Observe that

$$\left|\Pr_{\mathbf{G}_2}\left[\widetilde{b} = b\right] - \Pr_{\mathbf{G}_1}\left[\widetilde{b} = b\right]\right| = \mathsf{Adv}^{\mathsf{ROI}}_{\mathsf{KEM},A_2}$$

($\mathbf{G}_2$ corresponds to $b_{\mathsf{KEM}} = 1$, $\mathbf{G}_1$ to $b_{\mathsf{KEM}} = 0$).

A_3 attacks STREAM (Definition 10). First it generates $b \in \{0,1\}$ and bit strings K_{oracle} of length $\mathsf{KEM.OutLen}$ and $\mathfrak{K}_{\mathsf{oracle}}$ of length $\mathsf{KEM.CipherLen}$ uniformly at random. Then it runs A, playing the role of the key generation oracle (by using $\mathsf{KEM.KeyGen}$) and the role of the encryption oracle. Finally, when A outputs $\widetilde{b}$, A_2 outputs 1 if $\widetilde{b} = b$ and 0 otherwise. Observe that

$$\left|\Pr_{\mathbf{G}_3}\left[\widetilde{b} = b\right] - \Pr_{\mathbf{G}_2}\left[\widetilde{b} = b\right]\right| = \mathsf{Adv}^{\mathsf{ROI}}_{\mathsf{STREAM},A_3}$$

($\mathbf{G}_3$ corresponds to $b_{\mathsf{STR}} = 1$, $\mathbf{G}_2$ to $b_{\mathsf{STR}} = 0$) and clearly $\Pr_{\mathbf{G}_3}\left[\widetilde{b} = b\right] = \dfrac{1}{2}$.

Since b is uniformly random, by definition we have $\mathsf{Adv}^{\mathsf{ROI}}_{\mathsf{PKE},A} = 2 \cdot \left|\frac{1}{2} - \Pr_{\mathbf{G}_0}\left[\widetilde{b} = b\right]\right|$; putting all together, we obtain

$$\begin{aligned}\mathsf{Adv}^{\mathsf{ROI}}_{\mathsf{PKE},A} &= 2 \cdot \left|\sum_{1 \le i \le 3}\left(\Pr_{\mathbf{G}_i}\left[\widetilde{b} = b\right] - \Pr_{\mathbf{G}_{i-1}}\left[\widetilde{b} = b\right]\right)\right| \\ &\le 2 \cdot \left(\mathsf{Adv}^{\mathsf{ROR\text{-}CCA}}_{\mathsf{KEM},A_1} + \mathsf{Adv}^{\mathsf{ROI}}_{\mathsf{KEM},A_2} + \mathsf{Adv}^{\mathsf{ROI}}_{\mathsf{STREAM},A_3}\right).\end{aligned}$$