December 17, 2020

Analysis of the Butterfly Key Mechanism from IEEE P1609.2.1/D11

1 Introduction

We start with a brief overview of the Butterfly Key Mechanism protocol from IEEE P1609.2.1/D11. There are three types of parties involved: the end entity (EE), the cocoon key expander (CKE), and the butterfly key expander (BKE). An EE generates caterpillar secret keys for signing and encryption, along with the corresponding caterpillar public keys (for signing and encryption resp.). It also generates the secret key for the key expansion function. The public keys and the expansion function secret key are sent to the CKE via a secure channel. The CKE uses the expansion function and the expansion key to expand the caterpillar public keys into a set of cocoon public keys (for signing and encryption). These public keys are permuted and sent to the BKE. The BKE generates random offsets, which it uses to expand cocoon public signing keys into butterfly public signing keys. These butterfly signing keys are then certified by the BKE. The BKE also encrypts the offsets under the cocoon public encryption keys. The ciphertexts and certificates are returned to the CKE. The CKE forwards them to the EE who requested the keys and certificates. The EE computes the cocoon secret and public keys using the expansion function, decrypts the ciphertexts to obtain the offsets and computes the butterfly secret and public signing keys, and checks the certificates. The signing butterfly keys are for the ECDSA signature scheme used by EEs. The AES block cipher is used in the expansion function. The encryption keys are for the ECIES encryption scheme (used by the BKE to encrypt the offsets).

The analysis revealed that in the very strong threat model a slight tweak of the construction enables provable security for the security goals of end entities' (EEs') key secrecy and unforgeability of their signatures. For the goal of EEs' anonymity (privacy), the security provably holds if either cocoon key expander (CKE) or butterfly key expander (BKE) is honest (not corrupted), and if there is single BKE for each CKE. The analysis presented below details the modified protocol construction, provides proofs, and also analyzes the protocol under other, weaker threat models.

Abbreviations. Here we explain the abbreviations used throughout the report.

BKE	Butterfly key expander
CKE	Cocoon key expander
CDH	Computational Diffie-Hellman
DHIES	Encryption Scheme ECIES is based on
ECDSA	Signature scheme used in the protocol
ECIES	Encryption Scheme used in the protocol
EE	End entity
IND-CPA	Indistinguishability under chosen-plaintext attack
IND-CPA-RKA	Indistinguishability under chosen-plaintext and related-key attack
IND-CPA-wRKA	Indistinguishability under chosen-plaintext and weak related-key attack
ODH	Oracle Diffie-Hellman
PRF	Pseudorandom function
RKA	Related-key attack
RO	Random Oracle
SHA	Hash function used in the protocol
UF-CMA	Unforgeability under chosen-message attack
UF-RKA	Unforgeability under related-key attack
UF-wRKA	Unforgeability under weak related-key attack
ODH	Oracle Diffie-Hellman

2 Security Results

2.1 Scope and Assumptions

According to my conversations with Virendra Kumar, in my analysis I only studied the cryptographic core of the Butterfly Key Mechanism from Section 9 of the updated draft P1609.2.1/D11. In particular, I only focused on explicit certificates and the original mode with encryption and signature. I also made the following assumptions. The communication between EEs and CKE is private and authenticated, and the communication between CKE and BKE is authenticated. CKE verifies the EEs' enrollment certificates so that the attackers cannot impersonate EEs. The caterpillar keys of EEs are always honestly generated and thus have the right distribution and are independent from each other. Each EE is communicating with single CKE, and each CKE may talk to several BKEs.

2.2 Report Outline and Results Summary

The security results for the modified protocol depend on the threat model. Below we (I use terms "I" and "we" interchangeably in the report) treat different threat models separately. In Section 2.3 we consider the case when only one or two EEs are honest but the rest of the parties, including the BKE and CKE are corrupted. In this case, anonymity cannot hold, but unforgeability and key secrecy hold: Theorem 2.2 states the adversarial capabilities and goals, and under which computational assumptions security holds. Most of the assumptions are standard and widely used in proofs of cryptographic schemes, but one is not. In Section 2.4 we treat the case when the EEs and BKE are honest, but the rest are corrupted. In this case, unforgeability and key secrecy hold under the same computational assumptions as in the previous case. Anonymity holds assuming there is the only single BKE for honest EEs, or when there are several BKEs, but each BKE gets permuted requests containing equal portions of each EE's requests. There are only standard computational assumptions needed for this result. Theorems 2.4, 2.5 state the result in detail. In Section 2.5 the only honest parties are the EEs and and the CKE. Security results are as in the previous case. In Section 2.6 the attacker can corrupt BKE and CKE but it is passive (honest-but-curious). Anonymity cannot hold in this case, but unforgeability and key secrecy hold under the same assumptions as in the previous sections. Finally in Section 2.7 we assume that all parties are honest and the attacker can only observe the unencrypted communication. In this case, all security properties hold under the standard computational assumptions.

We now describe the analysis in detail.

2.3 Honest EEs

We start with the strongest security model, where one or two EEs, who talk to the same CKE, are honest, but every other party (including the CKE) are corrupted by an attacker. This is a very strong threat model, which may or may not be realistic in practice, but it is good to realize its impact on security.

In this case the attacker will know everything the CKE and BKEs know jointly, including the EEs' caterpillar, cocoon and butterfly public keys, the expansion functions and the private offsets BKEs choose, and also the BKEs' private certificate keys. We assume the attacker is active in that it can deviate from the protocol.

EE Anonymity (Privacy). Not surprisingly, the attacker can later link public keys and certificates to EEs (whether they belong to the same EE or not, and moreover, whose public key is whose). This is because the attacker will know the butterfly public keys of EE and the certificates, and to which EE the certificates are returned. Hence no EE anonymity can be achieved. And of course, the certificates cannot be trusted as they may have been produced by the attacker.

EE Key Secrecy and Signature Unforgeability. In practice one, of course, should be concerned about key secrecy. Another concern is unforgeability of signatures. These security goals are related. The private key recovery attack, if successful, would also imply a successful signature forgery attack, as signatures can

easily be forged using the recovered private key of the EE. However, the signature forgery attack, if successful, doesn't necessarily imply a successful private key recovery attack. So, protecting against signature forgery attacks is a strictly stronger security goal, as that would mean that the scheme is also protected against private key recovery attacks. Hence, we only explicitly focus on the goal of signature unforgeability, and not key secrecy.

It may seem that since the attacker does not know the private caterpillar keys of the honest EEs, the unforgeability holds. More precisely, one could expect that the signatures created by honest EEs are secure (unforgeable) given that a secure signature scheme is used. However, there is an issue here. Usually, a proof by reduction is used to prove statements like this. I.e., one proves that if an attacker can forge signatures on behalf of an honest EE in the big protocol, then there exists an attacker breaking security of the base signature scheme.

But such a proof by reduction will not immediately work here. This is because the standard (unforgeability under chosen-message attack or UF-CMA [5]) security of the base signature is not enough for the proof to go through. This is because the attacker against the protocol sees the caterpillar public key, computes the cocoon key, and can adversarially choose the butterfly offset that will influence the butterfly keys. Then, in order to win the attacker has to forge a signature under the resulting butterfly public key. And it does not seem to be possible to use this attacker attacking the protocol to create an attacker breaking the base signature scheme as the latter is dealing only with the single fixed key. What one would need is stronger security for the base signature scheme such as unforgeability under related-key attack (UF-RKA [3]). Such a notion requires unforgeability to hold even if the attacker can observe signatures under related keys.

Note that the above issue does not imply there is an attack on the protocol. It just shows that we do not have provable security guarantees given the known state-of-the-art.

Turns out, a weaker version of the unforgeability under related-key attack for the base signature scheme may work for us. But unfortunately, ECDSA signature scheme used in the protocol is not known to be provably secure in this sense, even though it is more likely to satisfy this weaker notion than resisting the general RKA attacks.

In order to obtain provable security under the weak RKA attacks (that we can cautiously conjecture) we need a very simple modification to the protocol. The change is as follows. In the current design the BKE picks the offset r, and sends it (encrypted) to EE via CKE, as before. But now $H(r||B_i)$ is used instead of r in butterfly key generation, where $||$ denotes concatenation and B_i is the cocoon public key of EE. Note that EE and BKE know B_i since CKE sends it to BKE. Here H is a cryptographic hash function with the range $\{0,1\}^\ell$, where ℓ is the bitlength of the private keys. In the protocol ℓ is 256, so either SHA-256 or SHA-3 with the output length of 256 bits would work. If ℓ is greater than the range of the hash, then H can be constructed from the hash by applying it to the same input with fixed and distinct prefixes.

The intuition for this modification is to prevent a corrupted BKE to choose r maliciously. Applying the hash makes the result look random despite the choice of r, as long as hash inputs do not repeat. The use of the cocoon public key is to prevent repeated inputs. Even if the malicious BKE chooses the same r, the cocoon keys will be distinct as the outputs of expansion functions will be distinct (with overwhelming probability).

We now provide a formal analysis.

First, we define the weaker notion of RKA security for discrete-log based signature schemes. We call it UF-wRKA.

Definition 2.1. [UF-wRKA] Fix a discrete-log based signature scheme. The experiment picks a random exponent x, which is the secret key and also picks random exponent offsets $a_1, ..., a_k$. The attacker is given the public key $x * g$, as well as the group description and the generator g, but it does not know x. In addition, the attacker is given all offsets $a_1, ..., a_k$, and also signing oracles $S_{x+a_1}(\cdot), \ldots, S_{x+a_k}(\cdot)$, which return signatures on messages of the attacker's choice computed under the corresponding secret keys (stated in the subscripts). Note that the attacker knows (can compute) the corresponding public keys too. The attacker wins if it can compute a signature on any message that is valid wrt any of the public keys g^{x+a_k}, given that it did not query the corresponding signing oracle on this message. The UF-wRKA advantage of the attacker is the probability of its winning. A signature scheme is called UF-wRKA secure if for any efficient (poly-time) attacker its UF-wRKA advantage is negligible.

Theorem 2.2. **[Unforgeability]** If at least one EE is honest, then no efficient attacker who can corrupt all other parties, including the CKE and BKE communicating with the EE (via CKE), learns the CKE's expansions for the EE $y_1, \ldots, y_k$ and can pick the BKE's offsets $r_1, \ldots, r_k$, and who observes the signatures created by the EE with its k resulting butterfly private keys, can forge a signature on behalf of the EE wrt to any of its butterfly keys. This assumes UF-wRKA security of the base discrete-log-based signature scheme, the block cipher used in the expansion functions being an ideal cipher, and that the hash function is modeled as a random oracle [4, 5].

It is extremely common to model hash functions as random oracles in the proofs. The AES is believed to be a PRF and also often assumed to be an ideal cipher, e.g., [6, 7, 9]. The ECDSA signature scheme used by the protocol uses discrete-log-type keys and is known to be UF-CMA assuming the hardness of discrete logarithm problem and that the underlying hash is a random oracle [8]. However, as we mentioned, the scheme is not known to be UF-CMA-wRKA, even though this seems plausible.

Proof. First we observe that even from the point of view of the malicious CKE, each EE's cocoon private key has the distribution of a sum of a randomly picked key a (unknown to the attacker) and a randomly picked CKE expansion y (known to the attacker). This is because the caterpillar private key a is honestly generated by the EE, and the expansion values y's, though known to the attacker, are unpredictable to it, as they are produced by honest EE. It is important here that each y is computed via a sequence of block ciphers computed under an honestly generated key on non-repeated inputs. This is a known construction of a pseudorandom generator, that is secure assuming the underlying block cipher is a pseudorandom function [5][1]. Note that XORing operations are not needed here, in that we would get the same security property if no XOR operations were used in addition to block cipher computations. Also note that even though security of the aforementioned block-cipher-based PRF relies on the secrecy of the private key, and in our case the attacker knows the key, the statement about the cocoon key distribution still holds since the attacker cannot select the inputs based on the key. But for this reason we will need to rely on a stronger, but still widely-used assumption of the block cihper being an ideal cipher, i.e., being a random permutation for every key.

Now we show that if there exists an efficient attacker $AdvA$ who can create a forgery on behalf of the honest EE, then we can construct an efficient UF-wRKA attacker $AdvB$ against the base signature scheme.

We construct $AdvB$, who uses $AdvA$ as a subroutine as follows.

$AdvB$ is given $X = x * g$, the group description, the generator g, random $a_1, ..., a_k$ and access to signing oracles $S_{x+a_1}(\cdot), \ldots, S_{x+a_k}(\cdot)$. To produce a forgery it will use the attacker $AdvA$.

Attacker $AdvA$, which attacks unforgeability of the protocol, expects to see the caterpillar public key $A = a * g$ and random (as justified above) expansion values $y_1, \ldots, y_k$. The attacker $AdvA$ makes random oracle queries on BKE's offsets $r_1, \ldots, r_k$ and expects to see the returned random values $z_1, \ldots, z_k$. It also expects access to the signing oracles $S_{a+y_1+z_1}(\cdot), \ldots, S_{a+y_k+z_k}(\cdot)$ that model signatures issued by the EE using its butterfly keys.

First $AdvB$ picks k random offsets $y_1, \ldots y_k$, and gives $AdvA$ the public key X and $y_1, \ldots y_k$. Whenever $AdvA$ makes the random oracle query $r_i||B_i$, $AdvB$ replies with $z_i = a_i - y_i$.

Whenever $AdvA$ makes a signing query M to its ith signing oracle, $AdvB$ sends M to its own ith signing oracle and forwards the reply to $AdvA$. When $AdvA$ outputs its forgery (a message and a signature), $AdvB$ outputs the same forgery.

We now claim that $AdvB$ wins whenever $AdvA$ wins. First, we observe that the simulation is perfect for $AdvA$ in that its view is like in the actual attack experiment. The EE's key it is given has the right distribution of a random key (because X given to $AdvB$ has the same distribution). The values y_i have the right distribution as per our argument in the beginning of the proof. Under the random oracle model, the BKE's offsets z's also have the right (uniform) distribution. It is important here that we assumed that $AdvA$ cannot make $r||B_i$ query repeat, as B_i depends on y_i. Now notice that the signatures for $AdvA$ are computed correctly by $AdvB$ because $AdvA$ expects signatures under keys $a + y_i + z_i$ which is exactly the private key underlying the corresponding $AdvB$'s signing oracle: $x + a_i = x + z_i + y_i = a + y_i + z_i$. Finally, the forgery of $AdvA$ is a valid forgery for $AdvB$. And clearly, if $AdvA$ is efficient, then $AdvB$ is efficient. □

[1] The result is part of the proof of security of the stateful counter mode encryption scheme.

2.4 Honest EEs and BKE

Now we assume that one or two EEs are honest, as well as one of BKEs they are related to. Every other party, including the CKE can be corrupted. The attacker will know the values we described in the Section 2.3 except for BKE's offsets and private key that it uses to issue certificates.

Unforgeability. The main difference here is that the attacker does not know the BKE's offsets. This is less information than in the threat model above (when BKE is corrupted), so the same security claim about unforgeability applies. And we still cannot prove security based on the standard notion of unforgeability for the base signature scheme ECDSA, and need to rely on its UF-wRKA security.

Anonymity. EE's anonymity depends on whether corrupted CKE deals with only one BKE or more.

If there are more than one BKE and EEs the CKE works with, then no anonymity holds as a malicious CKE can do the following attack. Say, the corrupted CKE gets requests from EE1 and EE2, and each is expanded into several cocoon keys using the expansion function. Then the CKE can send BKE1 all requests from EE1 and BKE2 all requests from EE2. Later, the attacker will not be able to tell the EEs' keys apart, but it will be able to tell their certificates apart, as they will be signed by different BKEs.

If there is only one BKE for each EE and CKE, then anonymity holds, because the attacker does not get any information about BKEs offsets due to encryption, and the offsets are random. Hence the resulting butterfly public keys look random to the attacker regardless of whether they come from the same or different EEs. To prove this we have to show that if the attacker can get any information about BKE's offsets, then one can break security of the base encryption or signature schemes. There is some complication with proving this as the attacker knows the CKE's expansion function key and hence the encryption key offset. However, as we showed in the proof of Theorem 2.2, the attacker still cannot influence them, as the expansion function key is chosen honestly by the EE, and the inputs are chosen independently from the CKE, and they don't collide. Hence, the output of the expansion function, which is the CKE's offset has the right uniform distribution assuming ideal-cipher security of the blockcipher used (XOR is not needed for this). Even though it's known to CKE, the proof still goes through.

To prove anonymity of the protocol we would like to use the fact that the underlying public-key encryption scheme, ECIES is secure. But again, the standard security (indistinguishability under chosen-plaintext attack or IND-CPA) is not immediately sufficient for us. Since the attacker breaking the protocol will see ciphertexts created under different but related cocoon keys (they are related because they correspond to the same caterpillar key and the attacker knows that key and the cocoon extensions), we need to rely on the security of ECIES under a weak notion of related-key attack. Again, ECIES is not known to be secure in this sense. The good news is that we can prove this (cf. Theorem 2.5).

Before we state our theorem, we define security of a symmetric encryption scheme against weak related-key attacks.

Definition 2.3. [**IND-CPA-wRKA**] Fix a discrete-log public-key encryption scheme. The experiment flips a random bit b, picks a random x, which is the secret key and also picks random offsets $a_1, ..., a_k$. The attacker is given the public key $x*g$, as well as the group description and the generator g, but it does not know x. In addition, the attacker is given all offsets $a_1, ..., a_k$. The attacker outputs k pairs $(m_0^1, m_1^1), \ldots, (m_0^k, m_1^k)$ of messages of the same length, in the message space of the scheme. It gets back k challenge ciphertexts $\mathcal{E}_{(x+a_1)*g}(m_b^1), \ldots, \mathcal{E}_{(x+a_k)*g}(m_b^k)$. The attacker has to output a bit and it wins if it is equal to b. The IND-CPA-wRKA advantage of the attacker is defined as 2 times its probability of winning minus 1. A scheme is called IND-CPA-wRKA secure if for any efficient (poly-time) attacker its IND-CPA-wRKA advantage is negligible.

Theorem 2.4. [**Anonymity**] If at least two EEs talking to the same CKE are honest, as well as **the only single** BKE they are related to, or when there are several BKEs, but each BKE gets permuted requests containing **equal portions** of each EE's requests, then no efficient attacker who can corrupt all other parties, including the CKE, learns the CKE's expansions y's for each EE, and who observes the ciphertexts created by the honest BKE, observes butterfly public keys of EEs and the BKE's certificates on them, can link these keys with the EEs. Linking here means telling if two keys belong to same EE or different EEs, and which

key belongs to which EE. This assumes the IND-CPA-wRKA security of the base encryption scheme, that the encryption scheme is robust [1] and uses discrete-log-type keys, the standard (UF-CMA) security of the signature scheme used by BKE, and the block cipher used in the expansion functions being an ideal cipher.

Theorem 2.5. [IND-CPA-wRKA Security of ECIES] The ECIES encryption scheme is IND-CPA-wRKA secure under the strong Diffie-Hellman assumption in the random oracle model, and assuming its base symmetric encryption and MAC are secure.

Note that ECIES is proven to be IND-CCA secure (a stronger notion than IND-CPA) under the same assumptions in [2]. I did not check if the standard implementation of ECIES complies with the general scheme (called DHIES) proven IND-CCA secure in [2]. In my analysis I will refer to DHIES.

Theorem 2.5 (proven later in this report) states IND-CPA-wRKA security of DHIES. Robustness mentioned in Theorem 2.4 means that it is hard to come up with a ciphertext that is valid under two different keys. In the base ECIES encryption scheme, if the same randomness is used by the sender, but the public encryption keys are different, then, assuming the underlying KDF function is a random oracle, the derived symmetric key for the MAC will be random and the probability that the second ECIES ciphertext will be valid is negligible, assuming that the MAC scheme underlying ECIES is UF-CMA. This is formally proven for DHIES in [1]. The ECDSA signature scheme used by the protocol uses discrete-log-type keys and is known to be UF-CMA assuming the hardness of discrete logarithm problem and that the underlying hash is a random oracle [8]. It is common to assume AES as an ideal cipher. So the protocol's anonymity is guaranteed under the well-accepted computational assumptions.

Proof of Theorem 2.4. First, just like in the proof of Theorem 2.2, we observe that even from the point of view of the malicious CKE, the EEs' cocoon private keys have the distribution of a sum of a randomly picked key (unknown to the attacker) plus a randomly picked key (known to the attacker). This is because a caterpillar private key a is honestly generated by the each EE, and the expansion values y_s, though known to the attacker, are unpredictable to it, as it is produced by the honest EE. This is because each y_s is computed via a sequence of block ciphers computed under an honestly generated key on non-repeated inputs. This is a known construction of a pseudorandom generator. Again, the XORing operations are not needed for security. Also note that even though security of the aforementioned block-cipher-based PRF relies on the secrecy of the private key, and in our case the attacker knows the key, the statement still holds since the attacker cannot select the inputs based on the key. But we assume that the block cipher is a random permutation for every key.

Now we note that if the attacker has no information about the BKE offsets, then all butterfly keys, of one or more EEs, will have the uniform distribution and hence will be unlinkable.

To model the security against the protocol attacker getting some information about the BKE offsets, we consider the following experiment associated with protocol attacker $AdvA$. The experiment flips a random bit b. $AdvA$ is given the EE's caterpillar public key $A = g^a$ and random $y_1, \ldots, y_k$ to model the unpredictable expansion values. (Just like in the proof of Theorem 2.2, $AdvA$ has to be given the secret key for the expansion function, but as we justified, it is equivalent to give $AdvA$ a random y's.) $AdvA$ is also given access to left-right encryption oracle that it can query on k message pairs $(m_0^1, m_1^1), \ldots, (m_0^k, m_1^k)$. The oracle returns encryptions of $m_b^1, \ldots, m_b^k$. The adversary outputs its guess d and wins if $d = b$. The advantage of the attacker is is defined as 2 times its probability of winning minus 1.

Now we show that if there exists an efficient attacker $AdvA$, then we can either construct an efficient IND-CPA-wRKA attacker $AdvB$ against the base public key encryption scheme or an efficient UF-CMA attacker against the signature scheme.

Let us consider 3 possibilities: when the attacker sends to BKE properly computed (via expansion) cocoon public keys of EEs and otherwise, follows the protocol, when at least on of the keys is not properly computed, and when the attacker sends to EE a ciphertext different than the BKE created.

1. We consider the case when the attacker $AdvA$ sends to BKE the properly computed (via expansion) cocoon public keys of EEs. In this case we construct $AdvB$, who attacks the security of the base public-key encryption scheme and uses $AdvA$ as a subroutine as follows.

   The IND-CPA-wRKA attacker $AdvB$ we construct is given a public key $X = g^x$ for an unknown x, as well as the group description and the generator g. In addition, the attacker is given all offsets $a_1, ..., a_k$.

$AdvB$ has to output k pairs of messages and later guess whether it's given encryptions of left or right messages. It will use the attacker $AdvA$.

The protocol attacker $AdvA$ expects the EE's caterpillar public key $A = a * G$ and random $y_1, \ldots, y_k$. To simulate these inputs, $AdvB$ gives $AdvA$ its own key X as $AdvA$ and offsets $a_1, ..., a_k$ as $y_1, \ldots, y_k$. When $AdvA$ queries k pairs $(m_0^1, m_1^1), \ldots, (m_0^k, m_1^k)$ of messages, $AdvB$ outputs the same messages and forwards the challenge ciphertexts to $AdvA$.

When $AdvA$ outputs its guess (left or right), $AdvB$ outputs the same guess.

We now claim that $AdvB$ wins whenever $AdvA$ wins. First, we observe that the simulation is perfect for $AdvA$ in that its view is like in the actual attack experiment. The EE's key it is given has the right distribution of a random key. The value y's have the right distribution as per our argument in the beginning of the proof. Now notice that the left-right encryption queries for $AdvA$ are computed correctly by $AdvB$ because $AdvA$ expects encryptions under key $(a + y_i) * g = (x + a_i) * g$, and this is exactly what $AdvB$ does. Finally, the correct guess of $AdvA$ is a correct for $AdvB$, and the advantages of the attackers are the same. And clearly, if $AdvA$ is efficient, then $AdvB$ is efficient.

2. Now we have to consider a possibility when a malicious CKE sends to BKE a maliciously created cocoon public key. In this case we cannot expect that the attacker learns no information about the offset BKE encrypts under such public key. But if the CKE forwards such a ciphertexts to the honest EE, it will decrypt the ciphertext using the distinct honest cocoon decryption key. And we are assuming the ciphertext will not be valid in this case, due to robustness.

3. Finally, if $AdvA$ tries to send to EE a ciphertext different than the BKE created, then the EE will detect this if the signature on the ciphertext will be invalid. If $AdvA$ manages to create a valid signature on a new ciphertext, then we can create an UF-CMA attacker for the signature scheme.

□

Proof of Theorem 2.5. It was shown in [2] that DHIES is IND-CCA under the strong Diffie-Hellman assumption in the RO model, and assuming the base symmetric encryption scheme and MAC are secure (Theorems 2 and 6 in the ePrint version of the paper `https://cseweb.ucsd.edu/~mihir/papers/dhaes.pdf`). Namely, Theorem 2 there shows that DHIES is secure under the oracle Diffie-Hellman assumption (ODH), and Theorem 6 shows that the ODH assumption is implied by the strong CDH assumption in the RO model. We extend the proof of Theorem 2 from [2] to show that DHIES is also IND-CPA-wRKA secure under the ODH assumption. We do not consider the chosen-ciphertext attacks and do not deal with the decryption oracles, because CCA attacks do not seem to be a threat in our setting.

We only modify Case 1 that deals with the ODH assumption and the case when the outputs of the hash function used in encryption do not look random. The other cases, that deal with security breaks for the case when the hash function outputs and hence symmetric keys look random, work for us without changes. In the proof for Case 1 one constructs the attacker $AdvB$ breaking the ODH assumption using the attacker breaking the standard IND-CCA security of DHIES. In our case, the IND-CPA-wRKA attacker will be used. The reduction is constructed in Figure 6 there. The ODH attacker C is given values $g, U = g^u, V = g^v, W$, where g is a random generator, u, v are random exponents and W is either $H(g^{uv})$ or a hash of a random group element[2]. The attacker has to figure out which case it is. In addition to other values, it is given oracle $O_v(\cdot)$ which on any input group element X returns $H(X^v)$. To compute the public key and the challenge ciphertext for the encryption attacker, the ODH attacker C uses V as the public key, U as the first "randomness" part of the DHIES ciphertext, and W to compute the keys for the symmetric encryption and MAC schemes used to encrypt the message m_b, where b is a random bit chosen by C and (m_0, m_1) are given by the DHIES attacker. The idea when the "real" W was used then the simulation is perfect, and when the "random" W is used, then random symmetric keys are used and security relies on underlying symmetric encryption and MAC.

[2]We use the multiplicative notation here following [2] unlike the additive notation usually used for elliptic curve groups, but this difference is not important for the security results.

The only difference is that in our case IND-CPA-wRKA attacker is given k random offsets, and instead of making one left-right query, the attacker now makes k, and gets k challenge ciphertexts, under different but related keys.

To make the reduction work for our case, we consider $k+1$ "hybrid" experiments associated with the protocol attacker $AdvA$. The experiments differ in how the k challenge ciphertexts are created. In experiment Exp_i ($0 \le i \le k$), in the first i ciphertexts the attacker is given, a random value is used instead of the output of the hash function. The rest of the ciphertexts are computed properly. So Case 1 described above considers the probability of the attacker detecting that at least one of the hashes underlying the challenge ciphertexts does not look random. In other words, the attacker can distinguish "real" ciphertexts from "random" ciphertexts, or Exp_0 from Exp_k. We claim that if the attacker can distinguish Exp_0 from Exp_k, then it must be able to distinguish Exp_i from Exp_{i+1} for some $0 \le i \le k-1$. And then we can provide a reduction and construct the ODH attacker.

We construct the ODH attacker $AdvB$ as follows. Recall that the ODH attacker C is given values $g, U = g^u, V = g^v, W$, where g is a random generator, u, v are random exponents and W is either $H(g^{uv})$ or a hash of a random group element. It is also given $O_v(\cdot)$. The attacker has to figure out which case it is.

$AdvB$ picks k random offsets $y_1, \ldots, y_k$. It gives $AdvA$ the EE's caterpillar public key $V \cdot g^{-y_i}$, and k cocoon public keys $V \cdot g^{y_1 - y_i}, \ldots, V, V \cdot g^{y_{i+1} - y_i}, \ldots, V \cdot g^{y_k - y_i}$. Now, to create challenge ciphertexts, $AdvB$ encrypts the first $i-1$ messages using randomly picked values in place of the hash, and the last $k-i$ messages properly according to the encryption algorithm using the corresponding keys. To encrypt ith message for $AdvA$, $AdvB$ uses its input U as the first part of the ciphertext, and then uses W in place of the underlying hash value to compute the symmetric keys. These keys are used then as usual according to the encryption algorithm.

Finally, $AdvB$ outputs the same bit as $AdvA$ does.

We claim that $AdvB$ wins if $AdvA$ wins. This is because the simulation of the two hybrid experiments is perfect, in that the view of $AdvA$ has the right distribution. The public keys $AdvA$ is given are $g^a = g^{v - y_i}$ and $g^{a+y_1}, \ldots, g^{a+y_k}$, and the ciphertexts are computed as they should in the experiments. This is clear for all ciphertexts except perhaps for the ith ciphertext. For that, note that W is either $H(g^{uv})$ or a random value. If the former, we get what's needed for Exp_i, and if not, then we get what's need for Exp_{i+1}. As usual with the proof involving hybrid experiments, there is a security loss proportional to the number of the hybrid experiments. Hence we get

$$\mathsf{Adv}^{\mathrm{IND-CPA-wRKA}}_{DHIES}(AdvA) \le k \cdot \mathsf{Adv}^{\mathrm{ODH}}(AdvB) \; .$$

Clearly, if $AdvA$ is efficient, then $AdvB$ is efficient.

The above implies that DHIES (and hence ECDIES) is IND-CPA-wRKA secure under the same assumptions as were used to prove the scheme IND-CCA secure, but security degrades linearly as k, the number of related keys, grows. □

2.5 Honest EEs and CKE

Now the attacker knows what is specified in Section 2.3 except for the expansion functions.

Unforgeability. Again, this is less information than in the threat model where only EE is honest (when CKE is corrupted), so the same security claim about unforgeability (Theorem 2.2) applies.

Anonymity. EEs' anonymity does hold assuming that honest CKE sends each BKE the permuted requests containing equal portion of each EE's cocoon keys or there is only one BKE for each EE, and since the cocoon keys have the right (uniform) distribution. The attacker knows the offsets BKE chooses, and the butterfly public keys, but it cannot connect them to EEs as it does not know the connection between cocoon public keys and EEs.

2.6 CKE and BKE are Passive (Honest-but-Curious)

In this case, the attacker can corrupt BKE and CKE but follows the protocol. In this case the attacker cannot create BKE's offsets maliciously and cannot send invalid keys and ciphertexts. But the attacker can still observe signatures created under related keys so we still need to rely on the UF-CMA-wRKA security of the signature scheme for unforgeability. Anonymity cannot hold for the same reason as explained in Section 2.3 for the case of active attacker.

2.7 All Parties are Honest, the Attacker is an Outsider

In this case, the attacker can only observe the communication sent via non-private channels. Then, both the EE anonymity/privacy and the signatures unforgeability hold based on the standard unforgeability property of the base signature scheme. Anonymity holds unconditionally given the assumption that the communication channel between EEs and CKE is private and all the attacker sees is the ciphertexts and certificates that do not contain links to EEs.

References

[1] Michel Abdalla, Mihir Bellare, and Gregory Neven. Robust encryption. In Daniele Micciancio, editor, *Theory of Cryptography*, pages 480–497, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[2] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The Oracle Diffie-Hellman assumptions and an analysis of DHIES. In David Naccache, editor, *Topics in Cryptology - CT-RSA 2001, The Cryptographer's Track at RSA Conference 2001, San Francisco, CA, USA, April 8-12, 2001, Proceedings*, volume 2020 of *Lecture Notes in Computer Science*, pages 143–158. Springer, 2001.

[3] Mihir Bellare, David Cash, and Rachel Miller. Cryptography secure against related-key attacks and tampering. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, pages 486–503, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[4] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[5] Mihir Bellare and Phillip Rogaway. Introduction to Modern Cryptography. 2005. `https://www.cs.ucdavis.edu/~rogaway/classes/227/spring05/book/main.pdf`.

[6] John Black, Phillip Rogaway, and Thomas Shrimpton. Black-box analysis of the block-cipher-based hash-function constructions from pgv. In Moti Yung, editor, *Advances in Cryptology — CRYPTO 2002*, pages 320–335, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[7] Anand Desai. The security of all-or-nothing encryption: Protecting against exhaustive key search. In Mihir Bellare, editor, *Advances in Cryptology — CRYPTO 2000*, pages 359–375, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[8] Manuel Fersch, Eike Kiltz, and Bertram Poettering. On the provable security of (EC)DSA signatures. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1651–1662. ACM, 2016.

[9] Éliane Jaulmes, Antoine Joux, and Frédéric Valette. On the security of randomized cbc-mac beyond the birthday paradox limit a new construction. In Joan Daemen and Vincent Rijmen, editors, *Fast Software Encryption*, pages 237–251, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.