

Enhanced Embarrassingly Parallel Attacks against EC-HMQV-C and ECMQV

Augustin P. SARR 

Laboratoire ACCA, UFR SAT, Université Gaston Berger, Saint-Louis, Sénégal
augustin-pathe.sarr@ugb.edu.sn

Abstract. In [15], Sarr introduced an embarrassingly parallel impersonation attack against ECMQV and ECHMQV-C whose core step is an adding-based pseudo-random walk over a cyclic group G , of order n , searching for a *decomposed i -point*. In this work, we revisit that construction in the richer multi-party setting where two sets of parties, the initiators $\{\hat{A}_1, \dots, \hat{A}_{z_1}\}$, and the responders $\{\hat{B}_1, \dots, \hat{B}_{z_2}\}$, interact under ECHMQV-C. First, we exhibit an embarrassingly parallel attack such that, when m processors are available, the attack runs in expected time

$$\frac{\sqrt{n}}{m z_1 z_2} (\text{costBA}(z_1) + \text{costBD}(z_1 z_2)),$$

wherein $\text{costBA}(z_1)$ is the time for a batch of z_1 elliptic-curve additions and $\text{costBD}(z_1 z_2)$ is the time for a batch of $z_1 z_2$ digest computations. Second, for any positive integer z_3 such that $z_1 z_2 z_3 \ll \sqrt{n}$, we propose an improved attack that runs in time

$$\frac{\sqrt{n}}{m z_1 z_2 z_3} (\text{costBA}(z_1 z_3) + \text{costBD}(z_1 z_2 z_3)),$$

Third, for ECMQV, assuming $z_1 z_3 \ll \sqrt{n}$, we propose an attack that runs in time

$$\frac{\sqrt{n}}{m z_1 z_3} \text{costBA}(z_1 z_3).$$

The attacks require no shared storage among processors, they require respectively $\mathcal{O}(z_1 z_2)$, $\mathcal{O}(z_1 z_2 z_3)$, and $\mathcal{O}(z_1 z_3)$ memory complexity at each processor, and only marginal server–processor communication.

As a concrete illustration, for a 128-bit ECHMQV-C instance, with $m = 2^{10}$ processors, $z_1 = z_2 = 2^{15}$, and $z_3 = 2^{10}$, our improved attack yields an effective parallel running time corresponding to 78 bits of security under an idealized massively parallel architecture in which the required batch elliptic-curve additions can be evaluated concurrently. This figure refers to parallel time rather than total computational work, which remains proportional to the number of elliptic-curve additions performed.

Keywords: embarrassingly parallel i -point search · impersonation attack · ECHMQV-C · ECMQV.

1 Introduction

The MQV protocol [10] and its elliptic-curve instantiation ECMQV are among the most widely standardised authenticated key-exchange schemes [1,2,3,7,8,12,20,21]. ECMQV

was deployed in validated industrial cryptographic products. Certicom’s Security Builder FIPS Java Module explicitly supported ECMQV as a key-agreement mechanism, while its Security Builder SSL toolkit provided ECMQV alongside ECDH and DH for commercial security applications [5,14,13,6]. ECMQV was also supported by the nShield family of hardware security modules [11]¹.

Krawczyk’s HMQV [9] provides a provably secure hashed variant. It has been shown in [17,18] that, without proper ephemeral public-key validation, HMQV-C may be vulnerable to key-compromise impersonation attacks. By ECHMQV-C, we refer to the three-pass elliptic-curve instantiation with public-key validation.

In [15], it was shown that both ECMQV and ECHMQV-C are vulnerable to an embarrassingly parallel impersonation attack. In ECMQV, the attacker searches for a *decomposed i -point* for the victim’s static public key A : a point $R \in G$ together with a scalar k such that $R = kP - (2^{\bar{1}} + i_0)A$ and $\bar{R} = (2^{\bar{1}} + i_0)$, where $\bar{R} = 2^{\bar{1}} + (x_R \bmod 2^{\bar{1}})$, wherein x_R is the x -coordinate of R and $\bar{1} = \lceil (\lceil \log_2 n \rceil + 1)/2 \rceil$. Using adding-based pseudo-random walks [19], each of the m processors iterates

$$u \leftarrow i_r(R), \quad k \leftarrow k + k_u \bmod n, \quad R \leftarrow R + W_u,$$

checking $\bar{R} = \bar{i}$ after each step. The expected total number of iterations across all m processors is $2^{\bar{1}}/m \approx \sqrt{n}/m$, already beating parallel Pollard rho [22] by approximately 25% and *requiring no shared storage*. In [15], for ECHMQV-C, it is additionally shown that when a server coordinates m processors and the attacker targets a single victim $\hat{A}$ to impersonate at any party $\hat{B}_j$ among z_2 parties $\hat{B}_1, \dots, \hat{B}_{z_2}$, the expected running time drops to $O(\sqrt{n}/(mz_2))$.

Our contributions. We generalize and refine the attacks presented in [15] to the *two-sided* multi-party setting. First, against ECHMQV-C, we consider an attacker who wishes to impersonate *some* party $\hat{A}_i$ (from a set of z_1 potential initiators) to *some* party $\hat{B}_j$ (from a set of z_2 potential responders). We exhibit a fully embarrassingly parallel attack achieving expected running time, when m processors are used,

$$\frac{\sqrt{n}}{mz_1z_2}(\text{costBA}(z_1) + \text{costBD}(z_1z_2)),$$

wherein $\text{costBA}(z_1)$ and $\text{costBD}(z_1z_2)$ are respectively the time for a batch of z_1 elliptic-curve additions and z_1z_2 digest computations. Second, we show that if z_3 is such that $z_1z_2z_3 \ll \sqrt{n}$, the attack can be improved so that when m processors are used, it runs in time

$$\frac{\sqrt{n}}{mz_1z_2z_3}(\text{costBA}(z_1z_3) + \text{costBD}(z_1z_2z_3)).$$

Third, for ECMQV, we show that, in a setting with z_1 initiators, when m processors are used, a decomposed i -point search can be performed with expected running time

$$\frac{\sqrt{n}}{mz_1z_3}\text{costBA}(z_1z_3),$$

¹ These references to past industrial deployments are provided for historical context only and do not constitute a security advisory regarding any currently deployed product.

as long as $z_1 z_3 \ll \sqrt{n}$. The attacks require no shared storage among processors, they require respectively $\mathcal{O}(z_1 z_2)$, $\mathcal{O}(z_1 z_2 z_3)$, and $\mathcal{O}(z_1 z_3)$ memory complexity at each processor, and no additional shared storage or inter-processor communication beyond that of [15].

The contribution of this work is not merely a direct application of the attack in [15] to a larger set of parties. The proposed construction introduces a two-sided multi-party search and an additional amplification dimension z_3 , and analyzes how these dimensions interact with massively parallel batch computation. We also extend the analysis to ECMQV, where the lack of an appropriate Random Oracle independence argument requires a separate heuristic treatment.

For illustration, when ECHMQV-C is instantiated over a group offering 128 bits of security, with $m = 2^{10}$ processors, $z_1 = z_2 = 2^{15}$, and $z_3 = 2^{10}$, the improved attack reduces the effective security level to 78 bits in terms of parallel running time. This estimate relies on an idealized massively parallel architecture in which the independent elliptic-curve additions within a batch can be evaluated concurrently; it should therefore be interpreted as a parallel-time complexity result, while the total computational work remains proportional to the 2^{40} additions performed in each batch.

This paper is organized as follows : In Section 2, we recall some notations and the EC-HMQV-C and ECMQV protocols. In Section 3, we present and analyze the enhanced attacks. In Section 4, we give some experimental results. We provide some concluding remarks in Section 5.

2 Preliminaries

For a finite set S , by $a \leftarrow_{\mathbf{r}} S$ we mean that a is chosen uniformly at random from S ; we write $a, b, c, \dots \leftarrow_{\mathbf{r}} S$ as a shorthand for independent uniform draws. By $\mathbb{F}_q$, we mean a finite field with q elements; $E(\mathbb{F}_q)$ denotes the set of rational points of an elliptic curve defined over $\mathbb{F}_q$. Throughout, P is a point of $E(\mathbb{F}_q)$ of large prime order n , $G = \langle P \rangle$ denotes the cyclic group generated by P , written additively. For $R \in G$, $\bar{R}$ is defined to be $(x_R \bmod 2^{\bar{l}}) + 2^{\bar{l}}$, where $\bar{l} = \lceil (\lceil \log_2 n \rceil + 1)/2 \rceil$. The cofactor $h = |E(\mathbb{F}_q)|/n$ is assumed small.

For $R \in G$, $i_r(R) = x_R \bmod 2^r$ for a fixed small constant r (e.g., $r = 5$). We write $\bar{H}$ for the l -bit output hash function used in ECHMQV-C; H is a hash function and MAC a message authentication code. We refer to the three-pass ECMQV protocol as ECMQV.

The ECHMQV-C Protocol. Let $(\hat{A}, a, A = aP)$ and $(\hat{B}, b, B = bP)$ be the identity-scalar-public-key triples of the initiator and responder, respectively. Protocol 1 describes a full execution of ECHMQV-C.

The main differences between ECMQV and ECHMQV-C are the following :

- In ECMQV, at lines 5 and 10, d_A and d_B are computed respectively as $d_A = \bar{X}$ and $d_B = \bar{Y}$;
- k_1 and k_2 are computed at lines 7 and 12 as $(k_1, k_2) = \text{KDF}(x_Z)$ where x_Z is the x -coordinate of Z ; and
- at lines 7, 12 13, and 16 the tags t_A and t_B are computed or verified as $t_A = \text{MAC}_{k_1}(3, \hat{A}, \hat{B}, X, Y)$ and $t_B = \text{MAC}_{k_1}(2, \hat{B}, \hat{A}, Y, X)$, respectively.

Algorithm 1 ECHMQV-C Key Exchange**The initiator $\hat{A}$ does the following:**

- 1 Choose $x \leftarrow_{\mathcal{R}} \{0, \dots, n-1\}$ and compute $X = xP$.
- 2 Send $(\hat{A}, \hat{B}, X)$ to $\hat{B}$.

At receipt of $(\hat{A}, \hat{B}, X)$, $\hat{B}$ does the following:

- 3 Validate the ephemeral public key X ; abort if validation fails.
- 4 Choose $y \leftarrow_{\mathcal{R}} \{0, \dots, n-1\}$ and compute $Y = yP$.
- 5 Compute $d = \bar{H}(X, \hat{B})$, $e = \bar{H}(Y, \hat{A})$, $s_B = (y + eb) \bmod n$, and $Z = h_{s_B}(X + dA)$.
- 6 Verify that $Z \neq \mathcal{O}$; abort if not.
- 7 Compute $k_1 = H(Z, 0)$, $k_2 = H(Z, 1)$, $t_B = \text{MAC}_{k_1}("1")$.
- 8 Send $(\hat{B}, \hat{A}, Y, t_B)$ to $\hat{A}$.

At receipt of $(\hat{B}, \hat{A}, Y, t_B)$, $\hat{A}$ does the following:

- 9 Validate the ephemeral public key Y ; abort if validation fails.
- 10 Compute $d = \bar{H}(X, \hat{B})$, $e = \bar{H}(Y, \hat{A})$, $s_A = (x + da) \bmod n$, and $Z = h_{s_A}(Y + eB)$.
- 11 Verify that $Z \neq \mathcal{O}$; abort if not.
- 12 Compute $k_1 = H(Z, 0)$, $k_2 = H(Z, 1)$.
- 13 Verify that $t_B = \text{MAC}_{k_1}("1")$; abort if not.
- 14 Compute $t_A = \text{MAC}_{k_1}("0")$.
- 15 Send $(\hat{A}, \hat{B}, t_A)$ to $\hat{B}$.

At receipt of $(\hat{A}, \hat{B}, t_A)$, $\hat{B}$:

- 16 Verify that $t_A = \text{MAC}_{k_1}("0")$; abort if not.

The shared session key is k_2 .**Impersonation Attacks using Decomposed i -point**

The following key observations, from [17,18,15], underpin the attacks we propose.

Lemma 1 (Impersonation using a decomposed ECHMQV-C i -point). *Let $\hat{A}$ be a party with static public key A (and corresponding private key a). If an adversary knows $R \in G$ and $k \in \{0, \dots, n-1\}$ such that $R = kP - i_0A$ and $\bar{H}(R, \hat{B}) = i_0$ for some $i_0 \in \{0, 1\}^1$ and some identity $\hat{B}$, then the adversary can impersonate $\hat{A}$ to $\hat{B}$ indefinitely (as in Attack 2 below), without knowledge of $\hat{A}$'s static private key a .*

Proof. When $\hat{B}$ receives $(\hat{A}, \hat{B}, R)$, it validates R and sends $(\hat{B}, \hat{A}, Y, t_B)$. It then derives a session and MACing key using the secret group element

$$Z = h_{s_B}(R + dA) = h_{s_B}(kP - i_0A + \bar{H}(R, \hat{B})A) = h_{s_B}(kP) = hk(Y + eB),$$

which the adversary, knowing k , computes identically and uses to forge t_A . $\square$

A pair (R, k) satisfying the conditions of Lemma 1 is called a *decomposed i -point* for A with respect to $\hat{B}$. One can observe that if an adversary has a decomposed i -point for A with respect to $\hat{B}$ and a *decomposed i -point* for B with respect to $\hat{A}$, then it can launch a man-in-the-middle attack, between $\hat{A}$ and $\hat{B}$ indefinitely.

The following lemma, geared to ECMQV, is similar to the previous.

Lemma 2 (Impersonation using a decomposed ECMQV i -point). *Let $\hat{A}$ be a party with static public key A (and corresponding private key a). If an adversary knows $R \in$*

Algorithm 2 Single-Victim Impersonation Attack [15]**Input:** A decomposed i -point (R, k) for A w.r.t. $\hat{B}$.

- 1 Send $(\hat{A}, \hat{B}, R)$ to $\hat{B}$.
- 2 On receipt of $(\hat{B}, \hat{A}, Y, t_B)$ from $\hat{B}$:
- 3 Compute $d = \bar{H}(R, \hat{B})$, $e = \bar{H}(Y, \hat{A})$, $Z = hk(Y + eB)$; verify $Z \neq \mathcal{O}$.
- 4 Compute $k_1 = H(Z, 0)$, $k_2 = H(Z, 1)$.
- 5 Verify $t_B = \text{MAC}_{k_1}("1")$; compute and send $t_A = \text{MAC}_{k_1}("0")$.
- 6 The shared session key is k_2 .

G and $k \in \{0, \dots, n-1\}$ such that $R = kP - (2^{\bar{l}} + i_0)A$ and $\bar{R} = 2^{\bar{l}} + i_0$ for some $i_0 \in \{0, 1\}^{\bar{l}}$, then the attacker can indefinitely impersonate $\hat{A}$ to any party, without knowledge of $\hat{A}$'s static private key.

3 Two-sided Multi-Party Embarrassingly Parallel Attacks

We consider two sets of parties $\{\hat{A}_1, \dots, \hat{A}_{z_1}\}$ and $\{\hat{B}_1, \dots, \hat{B}_{z_2}\}$; the static public key of a party $\hat{C}$ is denoted C . Note that we may have $z_1 = z_2$ and $\hat{A}_i = \hat{B}_i$. An attacker $\mathcal{A}$ aims to impersonate in ECHMQV-C some $\hat{A}_i$ to some $\hat{B}_j$. The computational resources available to $\mathcal{A}$ consist of a server S and m processors, each capable of performing batched elliptic-curve additions and digest computations.

The attack generalizes the search for a decomposed i -point of [15] by maintaining one current point R_i per $\hat{A}_i$ and checking all $z_1 z_2$ digests $\bar{H}(R_i, \hat{B}_j)$ at each step. Algorithm 3 describes the server's role; Algorithm 4 describes each processor's main loop.

Algorithm 3 Server execution against ECHMQV-C**Input:** $P, n, l, A_1, \dots, A_{z_1}, \hat{B}_1, \dots, \hat{B}_{z_2}$.

- 1 $i_0 \leftarrow_{\text{R}} \{0, 1\}^{\bar{l}}$.
- 2 **for** $j = 1$ **to** 2^r **do**
- 3 $k_j \leftarrow_{\text{R}} \{0, \dots, n-1\}$; $W_j \leftarrow k_j P$.
- 4 **end for**
- 5 Distribute $(P, i_0, n, A_1, \dots, A_{z_1}, \hat{B}_1, \dots, \hat{B}_{z_2}, k_1, \dots, k_{2^r}, W_1, \dots, W_{2^r})$ to each of the m processors and launch Algorithm 4.
- 6 **if** some processor returns $(i_0, k, R_i, A_i, \hat{B}_j)$ with $R_i = kP - i_0 A_i$ and $i_0 = \bar{H}(R_i, \hat{B}_j)$ **then**
- 7 Launch Attack 2 to impersonate $\hat{A}_i$ to $\hat{B}_j$ indefinitely.
- 8 **end if**

Proposition 1. Suppose a processor returns $(i_0, k, R_{i^*}, A_{i^*}, \hat{B}_{j^*})$. Then $R_{i^*} = kP - i_0 A_{i^*}$ and $\bar{H}(R_{i^*}, \hat{B}_{j^*}) = i_0$, so Attack 2 can be launched immediately.

Proof. At initialization (lines 2–4 of Algorithm 4), $R_i = \kappa_i P - i_0 A_i$. At each loop iteration, R_i is updated by $R_i \leftarrow R_i + W_u = R_i + k_u P$ and t is updated by $t \leftarrow t + k_u$.

Algorithm 4 Processor execution against ECHMQV-C

Input: $P, i_0, n, A_1, \dots, A_{z_1}, \hat{B}_1, \dots, \hat{B}_{z_2}, k_1, \dots, k_{2r}, W_1, \dots, W_{2r}.$

```

1 found  $\leftarrow$  false.
2 for  $i = 1$  to  $z_1$  do (batched)
3    $\kappa_i \leftarrow_{\mathbb{R}} \{0, \dots, n-1\}; R_i \leftarrow \kappa_i P - i_0 A_i.$ 
4 end for
5 for  $(i, j) \in \{1, \dots, z_1\} \times \{1, \dots, z_2\}$  do (batched)
6    $h_{ij} \leftarrow \bar{H}(R_i, \hat{B}_j).$ 
7   if  $i_0 = h_{ij}$  then found  $\leftarrow$  true.
8   end if
9 end for
10  $t \leftarrow 0.$ 
11 while found = false do
12    $u \leftarrow i_r(R_1).$   $\blacktriangleright$   $r$  least-significant bits of  $x_{R_1}$ 
13    $t \leftarrow t + k_u \bmod n.$ 
14   for  $i = 1$  to  $z_1$  do (batched)
15      $R_i \leftarrow R_i + W_u.$ 
16   end for
17   for  $(i, j) \in \{1, \dots, z_1\} \times \{1, \dots, z_2\}$  do (batched)
18      $h_{ij} \leftarrow \bar{H}(R_i, \hat{B}_j).$ 
19     if  $i_0 = h_{ij}$  then found  $\leftarrow$  true.
20     end if
21   end for
22 end while
23 Let  $(i^*, j^*)$  be such that  $i_0 = h_{i^* j^*}.$ 
24  $k \leftarrow t + \kappa_{i^*} \bmod n.$ 
25 Send  $(i_0, k, R_{i^*}, A_{i^*}, \hat{B}_{j^*})$  to  $S.$ 

```

Hence, after s iterations with walk steps $u_1, \dots, u_s,$

$$R_i = \kappa_i P + \left(\sum_{\ell=1}^s k_{u_\ell} \right) P - i_0 A_i = (t + \kappa_i) P - i_0 A_i = k P - i_0 A_i,$$

as claimed. The digest condition $\bar{H}(R_{i^*}, \hat{B}_{j^*}) = i_0$ holds by the check at line 11 (see also at line 19) of Algorithm 4. By Lemma 1, the attack is immediately applicable. $\square$

Theorem 1. Assuming the digest values $\bar{H}(R_i, \hat{B}_j)$ are uniformly and independently distributed over $\{0, 1\}^{\bar{l}}$, each iteration of Algorithm 4 succeeds with probability $p = z_1 z_2 / 2^{\bar{l}}$, independently of past iterations. Using m processors, the expected number of iterations per processor before a success is

$$\frac{2^{\bar{l}}}{m z_1 z_2},$$

and the expected total running time of the attack is

$$T = \frac{2^{\bar{l}}}{m z_1 z_2} (\text{costBA}(z_1) + \text{costBD}(z_1 z_2)) \approx \frac{\sqrt{n}}{m z_1 z_2} (\text{costBA}(z_1) + \text{costBD}(z_1 z_2)). \quad (1)$$

The attack has $\mathcal{O}(z_1 z_2)$ memory complexity at each processor.

Proof. At each iteration, the processor computes z_1 batched elliptic-curve additions (updating $R_1, \dots, R_{z_1}$) and $z_1 z_2$ batched digests (checking all pairs (i, j)). By the uniformity assumption, each digest h_{ij} equals i_0 with probability $1/2^{\bar{l}}$, and the $z_1 z_2$ events are independent (since the R_i are derived from independent random scalars, distinct public keys, or distinct responder identities). Hence the probability that at least one pair matches is

$$p = 1 - (1 - 2^{-\bar{l}})^{z_1 z_2} \approx \frac{z_1 z_2}{2^{\bar{l}}}$$

for $z_1 z_2 \ll 2^{\bar{l}}$. Each processor runs a Bernoulli process with success probability p ; by independence across m processors, the expected number of iterations until any processor succeeds is $1/(mp) \approx 2^{\bar{l}}/(m z_1 z_2)$. Each iteration costs $(\text{costBA}(z_1) + \text{costBD}(z_1 z_2))$, giving the claimed expected time. Since $\bar{l} = \lceil (\lfloor \log_2 n \rfloor + 1)/2 \rceil$, we have $2^{\bar{l}} \approx \sqrt{n}$, yielding the approximation. $\square$

Remark 1. – The assumption about the independence of the digest values $\bar{H}(R_i, \hat{B}_j)$, is implied by the Random Oracle model. We recall that the HMQV-C security arguments are provided under this assumption.

- The walk step indexes $u = i_r(R_1)$ are determined using R_1 alone; hence all processors can run fully independently with distinct random initialization, requiring no shared storage and no synchronization beyond the single-message report to the server.

Reducing the Number of Iterations through Larger Batches

In general, for any positive integers z_1 and z_3 , a single batch comprising $z_1 z_3$ ECC additions is more computationally efficient than z_3 sequential batches of z_1 ECC additions each. A similar observation holds for batched digest computations. An extreme case arises when each processor is equipped with dedicated co-processors capable of computing the $z_1 z_3$ ECC additions in parallel. We exploit this observation to reduce the expected number of iterations in our attack, at the expense of performing batched ECC additions and digest computations on larger sets.

To implement this strategy, at each processor, we initialize, for each public key A_i , not a unique starting point for the walk, but a vector of z_3 random starting points. The points are updated in each iteration (using a batch addition), and in each iteration the number of independent points to test grows to $z_1 z_2 z_3$. Assuming z_3 fixed and that $z_1 z_2 z_3 \ll 2^{\bar{l}}$ so that the approximation

$$(1 - 1/2^{\bar{l}})^{z_1 z_2 z_3} \approx 1 - z_1 z_2 z_3 / 2^{\bar{l}}$$

holds, the enhanced algorithm run by each processor is as Algorithm 5 (the server remains unchanged). We obtain Theorem 2, the proof is similar to that of Theorem 1.

Theorem 2. Assuming the digest values $\bar{H}(R_{il}, \hat{B}_j)$ are uniformly and independently distributed over $\{0, 1\}^{\bar{l}}$, each iteration of Algorithm 5 succeeds with probability $p = z_1 z_2 z_3 / 2^{\bar{l}}$,

Algorithm 5 Enhanced Processor execution against ECHMQV-C

Input: $P, i_0, n, A_1, \dots, A_{z_1}, \hat{B}_1, \dots, \hat{B}_{z_2}, k_1, \dots, k_{2r}, W_1, \dots, W_{2r}.$

```

1 found  $\leftarrow$  false.
2 for  $(i, l) \in \{1, \dots, z_1\} \times \{1, \dots, z_3\}$  do (batched)
3    $\kappa_{il} \leftarrow_{\mathcal{R}} \{0, \dots, n-1\}; \quad R_{il} \leftarrow \kappa_{il}P - i_0A_i.$ 
4 end for
5 for  $(i, j, l) \in \{1, \dots, z_1\} \times \{1, \dots, z_2\} \times \{1, \dots, z_3\}$  do (batched)
6    $h_{ijl} \leftarrow \bar{H}(R_{il}, \hat{B}_j).$ 
7   if  $i_0 = h_{ijl}$  then found  $\leftarrow$  true.
8   end if
9 end for
10  $t \leftarrow 0.$ 
11 while found = false do
12    $u \leftarrow i_r(R_{11}).$  ►  $r$  least-significant bits of  $x_{R_{11}}$  ( $i = 1$  and  $l = 1$ )
13    $t \leftarrow t + k_u \bmod n.$ 
14   for  $(i, l) \in \{1, \dots, z_1\} \times \{1, \dots, z_3\}$  do (batched)
15      $R_{il} \leftarrow R_{il} + W_u.$ 
16   end for
17   for  $(i, j, l) \in \{1, \dots, z_1\} \times \{1, \dots, z_2\} \times \{1, \dots, z_3\}$  do (batched)
18      $h_{ijl} \leftarrow \bar{H}(R_{il}, \hat{B}_j).$ 
19     if  $i_0 = h_{ijl}$  then found  $\leftarrow$  true.
20     end if
21   end for
22 end while
23 Let  $(i^*, j^*, l^*)$  be such that  $i_0 = h_{i^*j^*l^*}.$ 
24  $k \leftarrow t + \kappa_{i^*l^*} \bmod n.$ 
25 Send  $(i_0, k, R_{i^*l^*}, A_{i^*}, \hat{B}_{j^*})$  to  $S.$ 

```

independently of past iterations. Using m processors, the expected number of iterations per processor before a success is

$$\frac{2^{\bar{l}}}{m z_1 z_2 z_3},$$

and the expected total running time of the attack is

$$\begin{aligned} T &= \frac{2^{\bar{l}}}{m z_1 z_2 z_3} (\text{costBA}(z_1 z_3) + \text{costBD}(z_1 z_2 z_3)) \\ &\approx \frac{\sqrt{n}}{m z_1 z_2 z_3} (\text{costBA}(z_1 z_3) + \text{costBD}(z_1 z_2 z_3)). \end{aligned}$$

The attack has $\mathcal{O}(z_1 z_2 z_3)$ memory complexity at each processor.

Remark 2. For simplicity, we assume that all processors use the same value of z_3 . More generally, each processor may use a processor-specific value of z_3 according to its computational capabilities. This only requires minor adjustments to the analysis; the underlying ideas remain essentially unchanged.

An Improved Attack against ECMQV

In ECMQV, if $X, Y \in G$ are used as ephemeral keys in a session, the values of d and e are computed as $d = \bar{X}$ and $e = \bar{Y}$, we cannot then use the RO model to argue independence of the h_{il} in our analysis. So, if we attempt to adapt Algorithm 5 for ECMQV, we can no longer have all the pseudorandom walks next visited points determined solely by R_{11} . Each R_{il} determines the next visited point in the corresponding walk. Consequently, it no longer holds for batch additions that we use the same W_u . Fortunately, the W_u 's are drawn from a relatively small set (of size 2^r , we use $r = 5$). Thus, at steps 14–16, we can either perform: (a) 2^r batch additions, each involving the same W_u , or (b) a single batch addition involving different W_u 's. The best approach to use depends on z_1, z_3, r , and the specific characteristics of the implementation. We obtain the Algorithms 6 and 7; notice that each R_{il} is defined using an adding based pseudorandom walk with a random starting point.

Algorithm 6 Server execution against ECMQV

Input: $P, n, \bar{l}, A_1, \dots, A_{z_1}$.

```

1  $i_0 \leftarrow_{\mathcal{R}} \{0, 1\}^{\bar{l}}$ .
2 for  $j = 1$  to  $2^r$  do
3    $k_j \leftarrow_{\mathcal{R}} \{0, \dots, n-1\}$ ;  $W_j \leftarrow k_j P$ .
4 end for
5 Distribute  $(P, i_0, n, A_1, \dots, A_{z_1}, k_1, \dots, k_{2^r}, W_1, \dots, W_{2^r})$  to each of the  $m$  processors and launch Algorithm 7.
6 if some processor returns  $(i_0, \kappa_{i^*l^*}, R_{i^*l^*}, A_{i^*})$  with  $R_{i^*l^*} = \kappa_{i^*l^*} P - (2^{\bar{l}} + i_0)A_{i^*}^*$  and  $2^{\bar{l}} + i_0 = \bar{R}_{i^*l^*}$  then
7   Launch the adequate variant of Attack 2 to impersonate  $\hat{A}_i$  to any party indefinitely.
8 end if
```

Proposition 2 (Heuristic running-time estimate). *Assume that the pseudo-random walks generated by the attack can be modeled as random walks, and that the $\bar{l}$ least significant bits of the visited points are approximately uniformly distributed. If the successive iterations are treated as approximately independent, then an iteration succeeds at line 12 with probability*

$$p = \frac{z_1 z_3}{2^{\bar{l}}}.$$

Consequently, the expected number of iterations before the first success is

$$\frac{1}{p} = \frac{2^{\bar{l}}}{z_1 z_3} \approx \frac{\sqrt{n}}{z_1 z_3},$$

where the approximation assumes $2^{\bar{l}} \approx \sqrt{n}$.

With m processors, the corresponding expected running time is

$$T = \frac{2^{\bar{l}}}{m z_1 z_3} \text{costBA}(z_1 z_3) \approx \frac{\sqrt{n}}{m z_1 z_3} \text{costBA}(z_1 z_3).$$

The memory requirement is $\mathcal{O}(z_1 z_3)$ per processor.

Algorithm 7 Enhanced Processor execution against ECMQV

Input: $P, i_0, n, \bar{l}, A_1, \dots, A_{z_1}, k_1, \dots, k_{2^r}, W_1, \dots, W_{2^r}.$

```

1 found  $\leftarrow$  false.
2 for  $(i, l) \in \{1, \dots, z_1\} \times \{1, \dots, z_3\}$  do (batched)
3    $\kappa_{il} \leftarrow_{\mathcal{R}} \{0, \dots, n-1\};$   $R_{il} \leftarrow \kappa_{il}P - (2^{\bar{l}} + i_0)A_i.$ 
4   if  $2^{\bar{l}} + i_0 = \bar{R}_{il}$  then found  $\leftarrow$  true.
5   end if
6 end for
7 while found = false do
8   for  $(i, l) \in \{1, \dots, z_1\} \times \{1, \dots, z_3\}$  do (batched)
9      $j \leftarrow i_r(R_{il});$ 
10     $\kappa_{il} \leftarrow \kappa_{il} + k_j \bmod n;$ 
11     $R_{il} \leftarrow R_{il} + W_j;$ 
12    if  $2^{\bar{l}} + i_0 = \bar{R}_{il}$  then found  $\leftarrow$  true.
13    end if
14  end for
15 end while
16 Let  $(i^*, l^*)$  be such that  $2^{\bar{l}} + i_0 = \bar{R}_{i^*l^*}.$ 
17 Send  $(i_0, \kappa_{i^*l^*}, R_{i^*l^*}, A_{i^*})$  to  $S.$ 

```

Parallel implementation of batch additions and consequences

We stress that the term $\text{costBA}(z_1 z_3)$ depends on the underlying computational architecture. In particular, the $z_1 z_3$ elliptic-curve additions involved in a batch are mutually independent and can therefore be evaluated in parallel using dedicated arithmetic units or co-processors. In an idealized massively parallel architecture providing $z_1 z_3$ such units, the parallel cost of the batch is of the order of a single elliptic-curve addition, i.e.,

$$\text{costBA}(z_1 z_3) \approx \text{costBA}(1),$$

Consequently, for $z_1 = 2^{15}$, $z_3 = 2^{25}$, and $m = 2^{10}$, the resulting parallel running time is approximately $2^{78} \cdot \text{costBA}(1)$ for a 128-bit instance. This represents the idealized parallel-time complexity; the corresponding total computational work and hardware resources remain proportional to the 2^{40} additions performed in each batch.

It is also important to notice that in the idealized massively parallel architecture we described and the specific case $z_1 = 1$ (and $z_2 = 1$ for EC-HMQV-C), the attacks from [15] can be improved by any factor z_3 provided

$$\text{costBA}(z_3) \approx \text{costBA}(1) \text{ and } z_3 \ll \sqrt{n}.$$

The single-victim case $z_1 = z_2 = 1$ merits particular emphasis, as it combines the most realistic attack setting — a single, pre-identified target, rather than an indifferent choice among $z_1 z_2$ candidates — with the most defensible instance of the assumption $\text{costBA}(z_3) \approx \text{costBA}(1)$. When $z_1 = 1$, the z_3 batched walks operate on the same static public key A , with no data heterogeneity. The precise number of independent EC-addition units realistically available per device is architecture- and curve-size-dependent, and to our knowledge no published benchmark establishes this figure at the

scale $z_3 = 2^{25}$ assumed in the fully idealized example above. We simply note that any hardware-realistic z_3 — however many orders of magnitude below 2^{25} — still yields a proportional, non-negligible erosion of the effective parallel-time security level in the single-victim setting.

Remark 3. The decomposed i -point search against EC-HMQV can be adapted for EC-FHMQV, however, contrary to ECMQV and EC-HMQV-C

- the attack against EC-FHMQV have to be performed online (and not off-line), as the computations of d and e need the peer’s ephemeral key;
- and if the online attack succeeds, the decomposed i -point the attacker obtains can be used only once (in one session, and not indefinitely).

4 Some Experimental Results

To evaluate the experimental behavior of our algorithms, we implemented the proposed attack on elliptic curves with small parameters. We first generated 40, 50, and 60-bit prime-order curves in affine Weierstrass form $y^2 = x^3 + Ax + B$ over a prime field using the Magma computational algebra system. For multi-threaded experiments (simulating independent processors) and statistical measurements, we then ported the implementation to Java, using the Bouncy Castle ECC library [4]. Table 1 lists the curve parameters used. We conducted our experiments on a laptop equipped with an Intel Core i7-4600U processor at 2.7 GHz and 16 GB of RAM. For each curve size, we fixed $r = 5$ (then $2^r = 32$ different walk steps) and ran 100 independent trials to estimate the expected number of iterations.

Table 2 compares the empirical mean number of iterations with the theoretical prediction $2^{\bar{I}}/(mz_1z_2)$ for selected values of m , z_1 , and z_2 . To assess the agreement between the experimental results and the theoretical prediction, we compute the normalized quantity (NQ)

$$NQ = \frac{\text{MNoI} \times m \times z_1 \times z_2}{\sqrt{n}},$$

where MNoI denotes the mean number of iterations. Experimentally, this quantity remains close to 1 for all considered parameter settings, supporting the theoretical prediction of Theorem 1.

Table 1. Elliptic curve parameters for the two-sided i -point search.

Parameter	40-bit curve	50-bit curve	60-bit curve
p	1 099 511 627 791	1 125 899 906 842 679	1 152 921 504 606 847 009
A	776 203 427 801	309 080 485 083 429	442 101 273 648 441 055
B	1 060 406 378 309	42 332 728 083 522	606 815 451 238 597 808
P_x	281 540 352 699	922 731 844 497 576	207 978 671 727 726 979
P_y	982 324 199 637	420 806 730 728 823	714 773 102 348 778 908
$n = \text{ord}(P)$	1 099 512 384 147	1 125 899 962 050 597	1 152 921 503 838 574 936
h	1	1	1

Table 2. MNoI stands for Mean Number of Iterations (average over 100 trials). Empirical $NQ = \text{MNoI} \times m \times z_1 \times z_2 / \sqrt{n}$.

Curve	m	z_1	z_2	NQ
40-bit	64	32	4096	0.97475
40-bit	64	4096	32	0.92023
40-bit	128	32	4096	0.98985
40-bit	128	32	8192	0.88919
50-bit	32	512	1024	1.19621
50-bit	32	512	2048	0.95294
50-bit	32	1024	1024	1.07709
50-bit	64	256	1024	0.99824
50-bit	64	256	2048	0.99992
50-bit	64	512	1024	0.95630
60-bit	32	256	512	1.07080
60-bit	32	256	1024	0.99237
60-bit	64	128	128	1.01921
60-bit	64	128	256	1.07164

Curve	m	z_1	z_2	NQ
60-bit	64	128	512	0.96259
60-bit	64	128	1024	1.02928
60-bit	64	256	128	0.89359
60-bit	64	256	256	0.92023
60-bit	64	256	512	1.08632
60-bit	64	256	1024	1.28849
60-bit	64	512	128	1.12910
60-bit	64	512	256	0.99656
60-bit	64	512	512	1.15930
60-bit	64	512	1024	1.02341
60-bit	64	1024	128	1.00076
60-bit	64	1024	256	1.09219
60-bit	64	1024	512	1.18111

Table 3. MNoI stands for Mean Number of Iterations (average over 100 trials). Empirical $NQ = \text{MNoI} \times m \times z_1 \times z_2 \times z_3 / \sqrt{n}$.

Curve	m	z_1	z_2	z_3	NQ
40-bit	32	64	16	32	1.03683
40-bit	32	64	16	128	0.91603
40-bit	32	64	128	32	1.16014
40-bit	32	128	16	32	0.98754
40-bit	32	128	16	64	0.95378
50-bit	32	64	64	32	1.11149
50-bit	32	64	64	64	0.83466
50-bit	32	64	64	128	1.00998
50-bit	32	64	64	256	1.09051
50-bit	64	64	64	32	1.05864
50-bit	64	64	64	64	1.02341
50-bit	64	64	64	128	1.12742

Curve	m	z_1	z_2	z_3	NQ
60-bit	32	16	64	32	1.14735
60-bit	32	16	64	64	1.11023
60-bit	32	16	64	128	0.95210
60-bit	32	32	64	32	0.98901
60-bit	32	32	64	64	0.92945
60-bit	32	32	64	128	0.85647
60-bit	64	16	64	32	1.05990
60-bit	64	16	64	64	1.19537
60-bit	64	16	64	128	1.01250
60-bit	64	32	64	32	0.98272
60-bit	64	32	64	64	1.02508
60-bit	64	32	64	128	0.91603

In tables 3 and 4, we experimentally evaluate the normalized quantity

$$NQ = \frac{\text{MNoI} \times m \times z_1 \times z_2 \times z_3}{\sqrt{n}},$$

for different values of m , z_1 , z_2 (for EC-HMQV-C), and z_3 . As in the previous experiments, the resulting values remain close to 1, providing further experimental support for the theoretical complexity established in Theorems 1 and 2 and Proposition 2 (Indeed, Theorem 1 is a particular case of Theorem 2, with $z_3 = 1$).

Table 4. ECMQV MNoI stands for Mean Number of Iterations (average over 100 trials). Empirical $NQ = \text{MNoI} \times m \times z_1 \times z_3 / \sqrt{n}$.

Curve	m	z_1	z_3	NQ
50-bit	32	32	32	1.07380
50-bit	32	32	64	1.19630
50-bit	32	32	128	0.89169
50-bit	32	32	256	1.00534
50-bit	32	64	64	1.11121
50-bit	32	64	128	0.93918
50-bit	32	64	256	1.03017
50-bit	32	128	64	0.94696
50-bit	32	128	128	1.03829
50-bit	32	128	256	1.10530

5 Concluding Remarks

We have presented an enhanced embarrassingly parallel impersonation attacks against ECHMQV-C and ECMQV, that exploits the two-sided multi-party structure of realistic deployment settings. The attacks achieve expected running time

$$\frac{\sqrt{n}}{m z_1 z_2 z_3} (\text{costBA}(z_1 z_3) + \text{costBD}(z_1 z_2 z_3))$$

and

$$\frac{\sqrt{n}}{m z_1 z_3} \text{costBA}(z_1 z_3)$$

respectively, improving on the single-sided attack of [15] by an additional factor of $(z_1 z_3)$, wherein z_1 is the number of potential impersonation sources and z_3 an “amplification factor”, with the restriction $z_1 z_2 z_3 \ll \sqrt{n}$. The improvement is obtained at no extra cost in server storage or inter-processor communication, making the attack fully embarrassing parallel.

Natural countermeasures are (i) to increase the bit length of d and e to some value l' such that, even if a batch of 2^k ECC additions can be performed, the resulting $\mathcal{O}(2^{l'-k})$ computational cost remains out of reach or (ii) to compute d and e as in EC-FHMQV, so that an attacker cannot perform offline. Perhaps, an even stronger countermeasure is to combine the increase in the bit length of d and e with the computation of d and e as in EC-FHMQV [16,17,18], thereby requiring the attacker to perform the attack *online*.

References

1. ANSI X9.63 Standard: Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport using Elliptic Curve Cryptography. ANSI (2001).
2. Barker, E.B., Johnson, D., Smid, M.E.: Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography. NIST Special Publication 800-56A Rev. 1, National Institute of Standards and Technology (2007).

3. Barker, E.B., Chen, L., Roginsky, A., Vassilev, A., Davis, R.: Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography. NIST Special Publication 800-56A Rev. 3, National Institute of Standards and Technology (2018).
4. The Legion of the Bouncy Castle: Bouncy Castle Cryptography Library. <https://www.bouncycastle.org/>
5. Certicom Corp.: Security Builder FIPS Java Module. FIPS 140 validated cryptographic module (2008).
6. Certicom Corp.: Security Builder SSL. Product Documentation (2010).
7. IEEE 1363: Standard Specifications for Public Key Cryptography. IEEE (2000).
8. ISO/IEC IS 9798-3: Information Technology – Security Techniques: Cryptographic Techniques based on Elliptic Curves – Part 3: Key Establishment (2002).
9. Krawczyk, H.: HMQV: A high-performance secure Diffie-Hellman protocol. In: CRYPTO 2005. LNCS, vol. 3621, pp. 546–566. Springer, Berlin, Heidelberg (2005).
10. Law, L., Menezes, A., Qu, M., Solinas, J., Vanstone, S.: An efficient protocol for authenticated key agreement. *Des. Codes Cryptogr.* **28**(2), 119–134 (2003).
11. nCipher Security Limited: nShield F3 Level 2 Security Policy. FIPS 140 Cryptographic Module Validation Program, National Institute of Standards and Technology (2019).
12. NIST Special Publication 800-56A: Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography. NIST (2003).
13. NIST Cryptographic Module Validation Program: Security Builder FIPS Java Module. FIPS 140 Cryptographic Module Validation Program, Certificate #578, National Institute of Standards and Technology (2005).
14. NIST Cryptographic Module Validation Program: Security Builder FIPS Java Module. FIPS 140 Cryptographic Module Validation Program, Certificate #1056, National Institute of Standards and Technology (2008).
15. Sarr, A.P.: An Embarrassingly Parallel Cryptanalysis of EC(H)MQV. In *International Symposium on Foundations and Practice of Security* (pp. 305–315). Springer Nature Switzerland (2026).
16. Sarr, A.P., Elbaz-Vincent, P., Bajard, J.-C.: A secure and efficient authenticated Diffie-Hellman protocol. In: *EuroPKI 2009*. LNCS, vol. 6391, pp. 83–98. Springer (2010).
17. Sarr, A.P., Elbaz-Vincent, P.: On the security of the (F)HMQV protocol. In: *AFRICACRYPT 2016*. LNCS, vol. 9646, pp. 207–224. Springer (2016).
18. Sarr, A.P., Elbaz-Vincent, P.: On the separation between the FHMV and HMQV protocols. *Int. J. Appl. Cryptogr.* **3**(4), 377–393 (2017).
19. Teske, E.: On random walks for Pollard’s rho method. *Math. Comp.* **70**(234), 809–825 (2001).
20. Turner, S., Brown, D.L., Yiu, K., Housley, R., Polk, T.: Elliptic Curve Cryptography Subject Public Key Information. RFC 5480, IETF (2009).
21. Turner, S., Brown, D.L.: Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS). RFC 5753, IETF (2010).
22. van Oorschot, P.C., Wiener, M.J.: Parallel collision search with cryptanalytic applications. *J. Cryptol.* **12**(1), 1–28 (1999).