Speeding up elliptic computations for Ethereum Account Abstraction

Renaud Dubois

®Ledger
6 rue Gretry, 75002 Paris – France
firstname.lastname@ledger.fr
(**August 23, 2023**)

Abstract. Account Abstraction is a powerful feature that will transform today Web3 onboarding UX. This notes describes an EVM (Ethereum Virtual Machine) implementation of the well known secp256r1 and ed25519 curves [NIS23], optimized for the specificities of this EVM environment. Our optimizations rely on EVM dedicated XYZZ elliptic coordinates system, hacked precomputations, and assembly tricks to cut the over- all gas cost by a factor 5, reducing it from more than 1M to 200K/62K (with or without precomputations).
keywords:secp256r1, Secure enclave, FIDO2, WebAuthn, XYZZ coordinates, dedicated formulae, memory hack, precomputations, solidity, Pippenger/Shamir's trick

1 Introduction

Today, transactions on the Ethereum network are authenticated using the short Weierstrass curve secp256k1 [Bro10]. Account Abstraction allows users to utilize smart contract wallets that contain arbitrary verification logic instead of Externally Owned Accounts (EOAs) as their primary accounts. This feature is described in EIP4337 [BWG$^+$21] and has been implemented on the mainnet. It enables various use cases, including privacy-preserving applications, Vaults (using multisignatures/secret sharing protocols), aggregate signatures, and compatibility with existing signing mechanisms. The latter feature is particularly significant for improving the user experience. Currently, Web3 faces challenges due to a cumbersome user experience (UX), primarily concerning the protection of a user's seed through BIP39-like mechanisms [PRVB13], which may be unfamiliar to new users. The ability to implement any signature scheme to authenticate to a smart contract includes the utilization of traditional methods like WebAuthn [BBL$^+$21]/FIDO2. By employing such methods, Web2 users can be onboarded with a familiar UX, such as Face ID (Apple), Android fingerprint, or a security key implementing FIDO2, like a Yubikey or Ledger. All of these methods have in common the use of ECDSA as the allowed authentication algorithm over the secp256r1 curve. Unfortunately, using a non-native Ethereum curve results in a higher computational (i.e., gas) cost. While using the native curve incurs a transaction gas cost of 21K, replacing it with the currently available implementations leads to a 50x factor increase to 1M.

Our contribution The note is structured as follow:

– Section 2 provides a re-evaluation of the cost of elliptic curve representation systems based on EVM opcode costs. We propose a modified version of XYZZ coordinates, originally introduced by Sutherland, and demonstrate their optimality for specific use cases. We also highlight additional techniques to further reduce

the cost of point multiplication, which is the fundamental operation in most ECC applications.
- Section 3 describes how precomputations can accelerate ECC using the technique known as "Shamir's trick." We also discuss how the memory access rules in EVM render these optimizations ineffective and propose language hacks to restore their effectiveness.

The result of our work have been open sourced [Dub13] and should aid in integrating the WebAuthn mechanism into EVM chains.

1.1 WebAuthn, FIDO2.

FIDO is a phishing-resistant multi-factor authentication method based on established public key cryptography standards. It supports various hardware devices such as iPhones, Yubikeys, and Android phones. Web Authentication (WebAuthn) is a web-based API built upon FIDO, enabling websites to enhance their login pages with FIDO-based authentication on supported browsers and platforms. FIDO2 allows users to conveniently authenticate to online services using commonly available devices. The specific algorithm we focus on from the FIDO allowed list is ECDSA signature over secp256r1 (also known as P256).
Note: all results are directly applicable to ed25519, for simplicity sake the paper focus on secp256r1. Results for ed25519 are given in appendix 10.

1.2 Solidity, ECC and secp256r1.

Ethereum is a blockchain that introduced the concept of programmable smart contracts. Solidity (sol) is a high-level, object-oriented language used for implementing smart contracts that govern the behavior of accounts within the Ethereum state. Elliptic curve cryptography is an approach to public key cryptography based on the challenge of solving the discrete logarithm problem (DLP) over elliptic curves. It replaces the use of prime field groups $(\mathbb{F}_p, \times)$ with elliptic curve groups $(\mathbb{E}(\mathbb{F}_p), +)$. In most ECC systems, breaking the protocol can be reduced to solving the DLP. Therefore, scalar multiplication of a point is a crucial operation.

$$\begin{aligned} \texttt{ecmul} : \mathbb{Z} \times \mathbb{E}(\mathbb{F}_p) &\mapsto \mathbb{E}(\mathbb{F}_p) \\ (\lambda, G) &\mapsto Q = \lambda.G. \end{aligned}$$

It is also generally the most expensive computational part of the protocol. The EVM uses secp256k1 as native curve for the ECDSA authentication mechanism. While not being directly available as an opcode, a hacky use of the `ecrecover` operator (providing an ECDSA public key recovery function) makes point multiplication over secp256k1 relatively cheap [But18].

Most of existing implementations use either Projective or Jacobian coordinates to avoid the use of expensive divisions. While those formulae are optimized for classical architectures, the very specific weighting of EVM opcodes cost [Gla23] modifies the constraints for the choice of an optimal EVM solution. Table 1 provides the gas cost (which is related to the computational complexity of the opcode). A very unusual property is the fact that modular addition `addmod` and multiplication `mulmod` have

[1] Only alt_bn128 curve
[2] Only secp256k1 curve

Table 1. Arithmetic operations EVM opcodes [Gla23].

Group Operation	Notation	EVM opcode	EVM cost
$(\mathbb{F}_p, +)$	`modadd`	`addmod`	8
$(\mathbb{F}_p, \times)$	`modmul`	`mulmod`	8
$(\mathbb{E}(\mathbb{F}_p), 2)$	`ecDbl`	-	-
$(\mathbb{E}(\mathbb{F}_p)^2, +)$	`ecAdd`[1]	`ecAdd`	150
$(\mathbb{Z} \times \mathbb{E}(\mathbb{F}_p), \times)$	`ecmul`	`ecrecover`[2]	3000
$(\mathbb{Z} \times \mathbb{E}(\mathbb{F}_p), \times)$	`ecmulmuladd`	`ecrecover`	3000

the same cost. While elliptic operation formulae are optimized to reduce `mulmod` at maximum (considering constants multiplications and `addmod` negligible), this classic assumption doesn't hold in EVM. In the next section, ECC operations formulae are revisited according to this exceptional set of constraints.

2 Optimized formulae

2.1 Revisiting ECC with EVM opcodes constraints

Optimizing elliptic curve computations has been investigated from a long time by cryptographers. The reader is refered to [BL10] for a very documented and exhaustive database of formulae. Basically all computations are reduced to two function: ecadd which implements the law group of adding two points, and `ecdbl` which implements point doubling. When it comes to the double and add algorithm, in most cases the input points are given as normalized, ie with extra coordinates (z, zz) being equal to 1. Using this fact it is more efficient to use mixed coordinates using the operation `ecaddN`(P, Q), where P is given as normalized and Q as in the given projective (or jacobian) form. Tables 2 and 3 compare the 5 coordinates systems for short Weierstrass curves implementations. In such systems Specialized jacobian coordinates are the better performing. In the context of EVM, as shown in the tables, using specialized XYZZ coordinates instead save a 13% (resp 6.5%) for `ecaddN` (resp. `ecdbl`).

Table 2. Basic operations calls according to given system coordinates for `ecdbl`.

Operation Coordinates System	`#addmod`	`#mulmod`	`#sub`	`# mulcst`	Total (gas)
Projective	7	11	5	5	209
Specialized Projective	5	10	4	5	180
Jacobian	6	9	3	5	175
Specialized Jacobian	5	8	4	4	156
Twisted Edwards	8	8	4	1	156
Specialized XYZZ	5	9	2	3	146

The numbers provided assume the utilization of all the techniques described in this note (specialization, check removals, mixed coordinates). The libraries we examined employ a more direct implementation (as unnecessary checks are costly). The

[3] For Twisted Edwards, eccAdd and ecDbl are one function (complete formulae).

Table 3. Basic operations calls according to given system coordinates `ecaddN`.

Operation Coordinates System	`#addmod`	`#mulmod`	`#sub`	`# mulcst`	Total (gas)
Projective	6	11	5	5	169
Specialized Projective	6	11	5	1	169
Jacobian	6	11	6	1	174
Specialized Jacobian	6	11	6	1	174
Twisted Edwards [3]	8	8	4	1	156
Specialized XYZZ	6	10	3	1	151

disparity between the 'optimal' implementation in the target system and the actual implementations is presented in the Benchmarking section.

Note: while most modern implementations include Side Channel analysis countermeasures making CoZ coordinates or Twisted curves the most secure system, this threat is not addressed in the security objective, as the contracts solely implement verification.

2.2 Additional dedicated optimizations

Modular inversion over public data is usually performed using the extended euclidean algorithm. However, for prime field, it is possible to compute $a^{p-2} = a^{-1}$ instead. Although it is generally less efficient, the `modexp` precompiled contract perform this computation efficiently. Thus, it is used instead for all required modular inversions.

Reducing number of negations. While negation is almost cost-free in hardware, it incurs a cost of half a `mulmod` and shall be minimized. For example, by inspecting the output of the doubling algorithm, the output value (x, y) may be inverted to $(x, -y)$ for free simply exchanging the intermediate operators. The first operation in `ecãdd` involves inverting input y, so `ecdbl` can be tweaked to `ecdblneg` to avoid two subtractions. Since implementing $a-2b$ is more expensive using `mul` and `modsub` than `modmul` and `admod`, the constant $q-2$ (q the curve order) is used instead in various places.

Special case pruning. Prior to implement Shamir's trick, pruning point equality tests was considered. When the order is prime, looking at multiplication main loop of double and add algorithm, the special case `ecadd`(P, Q) with $P = Q$ cannot happen. Thus unnecessary checks may be removed. However, once using Shamir, it is possible to construct special cases that makes the `ecmulmuladd` fail.

2.3 Resulting formulae

XYZZ coordinates system is a specific representation of jacobian coordinates. In this system, a point over $\mathbb{E}(\mathbb{F}_p)$ is encoded as (x, y, zz, zzz) with $zz^3 = zzz^2$ and where its affine (X, Y) representation is obtained computing $X = \frac{x}{zz}$ and $Y = \frac{y}{zzz}$.

ecdblneg

```
input :$(x, y, zz, zzz) \in \mathbb{F}_p$
1 : $U = y$
2 : $V = U^2$
3 : $W = U * V$
4 : $S = x * V$
5 : $M' = 3 * (zz - X1) * (X1 + zz)$
6 : $x' = M^2 - 2 * S$
7 : $-y' = W * y + M' * (S - X3)$
8 : $zz' = V * zz$
9 : $zzz' = W * zzz$
10 : return $-2P = (x', -y', zz', zzz')$
```

ecnegaddN

```
input :$Qn : (X, Y), -P : (x1, -y1, zz, zzz) \in \mathbb{F}_p$
1 : $U2 = X2 * ZZ1$
2 : $S2 = y2 * zzz$
3 : $P = U2 - x1$
4 : $R = S2 - y1$
5 : $PP = P2$
6 : $PPP = P * PP$
7 : $Q = x1 * PP$
8 : $x' = R2 - PPP - 2 * Q$
9 : $y' = R * (Q - x3) - y1 * PPP$
10 : $zz' = zz * PP$
11 : $zzz' = zzz * PPP$
12 : return $R = P + Q = (x', y', zz', zzz')$
```

3 Pippenger/Straus/Shamir's trick

3.1 Description

In ECDSA, Schnorr and its variant, the verification process implies the computation of the addition of two point multiplication, refered later as `ecmulmuladd`:

$$\begin{aligned}\texttt{ecmulmuladd} : (\mathbb{Z} \times \mathbb{E}(\mathbb{F}_p))^2 &\mapsto \mathbb{E}(\mathbb{F}_p)\\ (\lambda, G) \times (\mu, Q) &\mapsto R = \lambda.G + \mu.Q\end{aligned}$$

As elliptic curves are noted additively, it is equivalent to a dual base exponentiation. This is a classic problem referred as product of powers or multibase exponentiations when the number of bases is any. This problem has been consecutively studied by Brauer (1939), Straus (1964) and Pippenger (1976) and later renamed as the 'Straus-Shamir's trick' when the number of bases is two. The basic trick consists in computing the sum P + Q, and then replacing the classic double and add algorithm by a single scan of the exponent chains. The number of ecmul sub operations drops from $(\frac{n}{2}\texttt{ecadd} + n\texttt{ecdbl})$ to $(\frac{3n}{4}\texttt{ecadd} + \frac{n}{2}\texttt{ecdbl})$ (n being scalar bitlength). The following algorithm describes the multi-input version (basic trick being an instance with k = 2).

StrausShamir

```
input :$P_1 \dots P_k \in \mathbb{E}(\mathbb{F}_p), (e_0 \dots e_k) \in \mathbb{F}_p$
1 : for $j = 0 \dots 2^{k-1}$ do   // Precomputations of all possible $P_i$ sums
2 :     $T[j] = \sum_{i=0}^{k} b_i(j) P_i$
3 : endfor
4 : for $j = l - 1 \dots 0$ do   // Scan scalars from MSB to LSB
5 :     $R = 2R$
6 :     $e = \sum_{i=0}^{k} e_i(j).2^i$
7 :     $R = R + T[e]$
8 : endfor
9 : return $R$
```

It is also possible to interleave the Shamir's trick with a windowing method, where k is the window size. When no precomputations is possible, interleaving Shamir with a windowing method. For a bitsize of 256 bits, dual base with $k = 4$ is a common choice (openssl, ours on starknet).

Window method

```
    input :$P \in \mathbb{E}(\mathbb{F}_p), \alpha = \sum_{i=0}^{l} \alpha_i.2^{i.k} \in \mathbb{N}$
1 : for $j = 1 \dots 2^{k-1}$ do   // Precomputations of all possible $\alpha_i.P_i$ sums
2 :     $T[j] = T[j-1] + P$
3 : endfor
4 : $R = T[\alpha_l]$
5 : for $j = l \dots 0$ do   // mainloop over window of $k$ bits
6 :     $R = 2^k.R$   // k successive ecdbl
7 :     $R = R + T[\alpha_i]$
8 : endfor
9 : return $R$
```

3.2 Precomputations.

When the input to `ecmulmuladd` is constant it is possible to improve algorithm 1 by externalizing the precomputations (step 3). For signature verification those constant are the base point P and public key Q. The basic trick doesn't require precomputations as it requires a single `ecadd`(P , Q) which costs less than the computations of 1 bit of the exponent. If precomputations are allowed, it is possible to improve the trick by increasing the number of inputs: the number of bases may be increased to w by providing

$$\{2^{\frac{j \cdot n}{k}} P_j, \forall j \in [1..k]\}.$$

Table 4. `ecmulmuladd` complexities in term of numbers of ec operations according to number of input $\#P_i$, $\#Q_i$ and window size w.

Implementation	$\#P_i$	$\#Q_i$	ω	Prec #`ecadd`	Prec	#`ecadd`	#`ecadd`
Naive	1	1	1	0	0	n	$2n$
Shamir-(2,1) [4]	1	1	1	1	64B	$\frac{3n}{4}$	n
Shamir-(2,2)	1	1	2	16	1KB	$\frac{15n}{16}$	$\frac{n}{2}$
Shamir-(2,8)[5]	4	4	1	768	16KB	n	$\frac{n}{8}$
Shamir-(8,1)[6]	4	4	1	768	16KB	$\frac{n}{4}$	$\frac{n}{4}$
Shamir-(2,4)	1	1	4	256	16KB	$\frac{n}{4}$	n

[4] Our implementation (1) choice
[5] alembichtech choice
[6] Our implementation (3) with prec. choice

3.3 Hacking EVM memory access cost.

The use of precomputations requires the use of large arrays of elliptic points. Unfortunately it is not possible to declare arrays of constant in solidity. The cost of access to storage as depicted in Table 5 would cost 2100 for each first access to a cell during verification. For a $k = 8$ multibase evaluation, the average number of cold access is around 50. This already has a cost of $100K$ gas. This kills the expected gain. To get around this limitation, contracts like `sstore2.sol` wraps the `extcodecopy`, deploying the given table as a contract to access instead of a storage array. The cold access with `extcodecopy` is paid only once at first access to the contract. We then devised a way to use the cheapest `codecopy` instruction to access a given static array T:

1. declare a `string constant` with a magic value, of same size as the precomputations,
2. declare a function that simply return the constant,
3. to compute the contract of a user, parse the dummy contract until magic value, then override it with the precomputations,
4. deploy the overriden bytecode.

Note that this trick enables to use array of constant in solidity with a x33 more efficient cost than `sstore2.sol`.

Table 5. Access costs according to memory type in the EVM.

Instruction	Memory Type	Cold Access	Warm Access
`sload`	Storage	2100 [7]	100
`extcodecopy`	External code	2100 [8]	100
`codecopy`	Internal code	3	3
`mload`	Internal memory	3	3

4 Implementation details

Arithmetic. The code has been optimized for Weierstrass curves with coefficient $a = -3$. Consequently, it can be easily adjusted and all secp curves. By employing isogeny, the code can be adapted to any Weierstrass curve with a non-zero value for a. The coordinates system used is XYZZ , as previously described, employing the negation trick.

Tradeoff. Two versions are available: the first one utilizes a Shamir's trick with P and Q as bases. The second one performs precomputations of the 8 values $P_i = 2^{64*i}.P, Q_i = 2^{64*i}.P, \forall i \in [1..8]$. The precomputations are conducted off-chain using js or sagemath.

From solidity to inlined assembly. As for most implementation aiming for efficiency, assembly is necessary. The complete Shamir's trick has been implemented in `asm`. The code is heavily inlined, as initial benchmarks indicate a 20% improvement simply by inlining `ecadd` and `ecdbl`.

[7] Once per cell
[8] Once per contract

5 Benchmarking

This section provides the benchmark for several solidity ECC libraries. To distinguish the benefits obtained from algorithmic optimizations and language optimizations, The measurements were performed in the hardhat and forge environment using same number of runs (degree of optimization in the solidity compiler).

Note: if any error regarding credits or library names are present, please contact the author for a prompt update and apologies.

5.1 ECC solidity libraries

Table 6 outlines the algorithmic choices of six solidity libraries (including our own). It presents : the size of precomputations, the system coordinates used, the existence of mixed coordinates, specialized coordinates, and utilization of Shamir's trick (sorted by columns).

Table 6. Existing libraries characteristics.

Library	asm	prec.	coordinates	Mixed	specialized	Shamir's trick	Link
orbs network	×	0	proj.	×	×	×	orbs-network
alembich-tech	×	16 KB	proj.	×	×	✓	alembich
Numerology [9]	×	×	jacobian	✓	✓	×	Numerology
Maxrobot	✓	×	proj.	×	✓	×	maxrobot
Androlo	×	×	jacobian.	✓	×	×	Androlo
itsobvioustech	✓	0	modified jac.	×	✓	✓	itsobvioustech
Ours(1)	✓	0	XYZZ	✓	✓	✓	[Dub13]
Ours(2)		128B	XYZZ	✓	✓	✓	
Ours(3)		16 KB	XYZZ	✓	✓	✓	

Table 7 provides the actual number of basic operations used to implement `ecadd`.

Table 7. Actual number of opcodes in `ecadd` solidity implementations.

Operation	addmod	add	mulmod	sub	mulcst	Total
Ours	6	0	10	3	1	**151**
Androlo	6	0	13	5	0	167
orbs-network alembich-tech	6	0	14	4	0	172
Numerology	6	0	16	5	1	209
itsobioustech	3	5	19	5	1	214

5.2 Practical Results

While the previous subsection provides an insight into the expected asymptotic gain, accurately predicting the exact gain is challenging due to various additional factors involved:

[9] Numerology is highly optimized with GLV for secp256k1

- the cost of handling extra coordinate,
- the cost of memory access,
- extra hidden instructions (e.g., `push, pop`),

The gas cost was measured using the forge environment with an optimizer set to a range of steps from 10^4 to 10^5. Solidity version is `0.8.20`. The only modifications performed on source library were done to update solidity compatibility. As WebAuthn is not implemented for all libs, the same implementation with a $40K$ cost is applied generically. Table 8 provides the gas cost for all given libraries. Atomic functions `eccAdd` and `ecDbl` costs are provided to separate the different gains. Upon examining the results the following statements can be made:

1. Using assembly provides a larger speed up than improved formulae.
2. Inlining is crucial in solidity (25%gain), the compiler is not as effective as a gcc counterpart for example. This leads to a less readable code.
3. Using a larger number of base than 8 is ineffective, as the overhead of handling masks to compute the element to access approached one third of the `ecmulmuladd` operation.

Table 8. Practical gas cost measurement using forge.

Library	ecaddN	ecDbl	ecmulmul (ecdsa)	WebAuthn (full)	Deployment (contract)	Deployment (precomputations)
orbs-network	2250	1750	1.06M	1.1M	375K	0
Androlo	2073	1229	866K	906K		0
Maxrobot	1949	1502	760K	790K		0
Numerology	1973	1003	422K	462K		0
alembich-tech	2250	1750	335K	375K	2M	3.2M
itsobvioustech	946	578	290K	330K	590K	0
Ours(1)	566[10]	522	**202K**	242K	1.03M	0
Ours(3)			**69.1 K**[11]	115K	713K	3.2M

Amortization of precomputations. The additional cost of deploying a 16KB contract (3.2M) is compensated after 30 transactions. Note that this cost could be divided by 2 using a wNAF like approach, at the expense of a little extra computations.

EIP-4844 [BD21] and calldata reduction. When EIP4844 is adopted, with a calldata cost per byte of 3, passing precomputations as calldata would cost 48K. It will provide a new tradeoff, avoiding the extra contract deployment in exchange of this fee plus an integrity verification of the table.

5.3 Testing, Fuzzing

The Wycheproof 9 project is a framework providing many edge cases for cryptographic protocols, including secp256r1. Table 9 provides the results of the extended Wycheproof tests that are run against the libraries. Some of the test related to integer length were disabled as by essence an `uint256` EVM integer cannot handle number larger than 2^{256}.

[10] Not inlined in main loop
[11] Using the hackmem trick

Table 9. Detected anomalies on target libraries.

Library	orbs-network	alembich	Numerology	Androlo	Maxrobot	itsobvioustech	Ours
Detected errors		Malleability			**Null sig** Duplication	**Null sig** Duplication	

– Duplication refers to an error occurring when the operation ecadd is called on the same point. Wycheproof doesn't detect the itsobvioustech duplication problem, as it is hidden by Shamir's trick. Theoretically, a dishonest user could exploit this flaw to forge invalid signatures that could be accepted or valid signatures refused. However it doesn't seems possible to forge such vector without the private key knowledge, which should reduce the threat to double spend under very specific assumptions. The impact is low but shall be corrected.
– Null signature is a critical flaw, as it allows to submit any Tx with a valid signatures, potentially stealing all coins of the user. The authors have been warned and PR submitted to obvious and maxrobot.
– The signature malleability is not compliant to ECDSA specification. It is possible to submit $(r, s + q)$, q being the curve order as a valid signature. While no direct exploit appears (except use requiring Strong Unforgeability property like Mtgox) it shall be avoided.

6 Conclusion

This notes presents an optimized implementation of ECC computations in the EVM. The implementation incorporates algorithmic optimizations specifically tailored to meet EVM constraints, careful memory access considerations, and precomputations, resulting in a performance approximately six times faster than previous best one. This optimized implementation facilitates the testing of Webauthn authentication within smart contracts. While the integration of dedicated opcodes is under discussion, we believe that this implementation can contribute to the discussions and offer a practical approach for its implementation in most application chains, including ETH, based on its usage. Account Abstraction is a powerful tool that simplifies the onboarding process for the next billion users. By combining traditional tools with EIP4337, it becomes possible to make the utilization of Web3 as frictionless as Web2, minimizing the differences between the two worlds to self-custody.

Further work

1. implement a tradeoff with 12 bases incorporating two additions (one with a point of T_p, one of T_q). This tradeoff would reduce the deployment cost by 75%, while maintaining equivalent performances. (refer to the algorithm described in appendix).
2. secp256k1. Some of the methods described here are applicable to secp256k1. While the use of hacky mul [But18] enables very efficient Schnorr algorithm implementation, it may be still necessary to have optimized implementation of ecadd as provided by Numerology library.
3. Schnorr/EdDSA. For now, only a limited set of FIDO devices support EDDSA which is a Schnorr signature. Schnorr allows easiest integration of MPC signatures

([NRS21]) and partial aggregation. Using the Twisted Edwards coordinates would have a very limited impact on the speed (foresight around 3%).

4. ZkEVM bench. The benchmarks provided assumed an identical rating of the memory and opcodes. The final choice (with/without precomputations) and results may differ if the opcodes cost is modified (for instance by their zkprovability friendliness or storage cost).
5. non EVM chains. While this note addresses EVM and zkEVM chains as most promising technology, some of the results described here are available in other frameworks. A FCL ecmulmuladd implementation is already integrated in Braavos wallet over Starknet . We plan to introduce the precomputational version next to obtain the fastest Starknet P256 signer.

Acknowledgments. We would like to express our gratitude to qdqd, the founder of Ledger's solidity guild, for his valuable teaching, kind explanations, and advices on Solidity. We also thank Yannick Seurin and Victor Servant for useful discussions and comments.

References

BBL+21. John Bradley, Christiaan Brand, Adam Langley, Giridhar Mandyam, Nina Satragno, Nick Steele, Jiewen Tan, Shane Weeden, Mike West, and Jeffrey Yasskin. web authentication:an api for accessing public key credentials level 2. FIDO Association, 2021. `https://www.w3.org/TR/webauthn-2/`.

BD21. Vitalik Buterin and Ansgar Dietrichs. "EIP-4488: Transaction calldata gas cost reduction with total calldata limit, 11/2021. `https://eips.ethereum.org/EIPS/eip-4488`.

BL10. Daniel Bernstein and Tania Lange. Explicit elliptic formulas database. NIST, 2010. `//https://https://hyperelliptic.org/EFD/l..`

Bro10. Dan Brown. SEC2 : Recommended elliptic curve domain parameters. Certicom research, 01/2010. `https://www.secg.org/sec2-v2.pdf`.

But18. Vitalik Buterin. you can *kinda* abuse ecrecover to do ecmul in secp256k1 today. GitHub, 2018. `https://ethresear.ch/t/you-can-kinda-abuse-ecrecover-to-do-ecmul-in-secp256k1-today/2384`.

BWG+21. Vitalik Buterin, Yoav Weiss, Kristof Gazso, Namra Patel, Dror Tirosh, Shahaf Nacson, , and Tjaden Hess. ERC-4337: Account Abstraction using alt mempool [draft]. Ethereum Improvement Proposals, no. 4337, 09/2021. `https://https://eips.ethereum.org/EIPS/eip-4337`.

Dub13. Renaud Dubois. fresh crypto lib, a cryptographic library for blockchain uses. GitHub, 10/2013. `https://github.com/rdubois-crypto/FreshCryptoLib/blob/master/solidity/FCL_elliptic.sol`.

Gla23. Gray Glacier. an ethereum virtual machine opcodes interactive reference. GitHub, 2023. `https://www.evm.codes/?fork=grayGlacier..`

NIS23. NIST. Digital signature standard (DSS, fips186-5. NIST, 2023. `https://https://csrc.nist.gov/publications/detail/fips/186/5/final`.

NRS21. Jonas Nick, Tim Ruffing, and Yannick Seurin. Musig2: Simple two-round schnorr multi-signatures. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part I*, volume 12825 of *Lecture Notes in Computer Science*, pages 189–221. Springer, 2021.

PRVB13. VMarek Palatinus, Pavol Rusnak, Aaron Voisine, and Sean Bowe. Mnemonic code forgenerating deterministic keys. EBitcoin Improvment Proposal 39, 10/2013. `//github.com/bitcoin/bips/blob/master/bip-0039.mediawiki..`

A Appendix

A.1 Ed25519

The FreshCryptoLib also implements `mulmuladd` with basic trick for the ed25519 curve. As expected results are very close to secp256r1.

Table 10. ed25519 implementation.

Implementation	#Bases	ecmulmul (measured)	Deployment (Prec.)	Comment asm+basic trick
Ours	2	206K	0	
ed25519-solidity	1	1.25M	0	no asm, no trick

A.2 Further tradeoffs

Table 11 compares the number of basic operations of current implementations with futures. It also provides some estimation over the expected resulting gas cost of those implementations. It is assumed that the target curve has a 256 bits modulus. Those projections shall be taken with cautious as we tried to integrate the increased complexity of mask computations and hybrid memory access.

Splitting. It is possible to reduce the number of precomputations at the expense of extra addition in the multibase exponentiation. For instance it is possible to compute separately all the possible sums of multiple of G of the base on one side, and multiples of Q on the other side. A single look-up table to compute a sum of bases is replaced by two look up and additions in the step 6 of Straus-Shamir. The multiple of G being shared by all users, this table deployment is paid only once.

Windowing/NAF. The last implementation shall interleaved a 1-NAF computation with a 4 bases Straus-Shamir. It shall take the value $2^{128}Q$ and $(2^{128}+1)Q$ in calldata and $2^{128}G$ as a contract constant.

Table 11. Estimation of alternative tradeoffs implementations.

Implementation	#Bases	ecdbl	ecadd	ecmulmul (measured)	Deployment (Prec.)	Comment
Ours(1)	2	256	192	201K	0	
Ours(3)	8	64	64	61.6K	3.2M	
				(estimated)		
Future(1)	8	64	32	130K	200K	1+1Kb of precomputations
Future(2)	12	43	86	85K	800K	4+4Kb of precomputations
Future(3)	4	128	128	160K	0	$2^{128}Q$ in calldata