Agile Asymmetric Cryptography and the Case for Finite Fields

Anna M. Johnston
Juniper Networks

2024/09/11

Abstract

Cryptographic agility, the ability to easily and quickly modify cryptography in a system, is one of the most important features of any cryptographic system. Any algorithm may be attacked and, at some point in time, be broken. The most obvious solution is to change the cryptographic algorithm, however this has high risk and cost. Another solution is to use agile algorithms. Agile algorithms have security parameters easily changed to increase protection against attacks.

In this paper we will show that finite field based algorithms are the most agile of currently used classical cryptography. A critical portion of this will be to show that the bottleneck for the primary costing attack, the number field sieve, is the linear algebra portion of the attack, and not the sieving portion.

This paper examines the agility of all three algorithm categories and dispels a few myths about their strengths.

1 Background

Cryptographic agility should be a key feature of any cryptographic system. All algorithms may be attacked and, at some point in time, be broken. Systems must adapt to keep pace with improved attacks or compute power. This is true for both symmetric and asymmetric (public key) algorithms. For example, the old **D**ata **E**ncryption **S**tandard, DES, was never analytically broken. The key length was short and compute power grew to threaten it. More analytic progress has been made against AES (**A**dvanced **E**ncryption **S**tandard), the replacement for DES, but key sizes are much larger and the analytic progress has not led to any practical attacks. Furthermore, AES has multiple key sizes which allow security to be increased. There are two options to protect data in this ever changing landscape: create new

algorithms to replace those which show weakness or use better security parameters. Both options have issues.

New algorithms have, by their shorter time on the analysis table, a higher probability of being broken. The more analysis an algorithm receives, the higher the confidence in its strength. Furthermore, complications hide weaknesses. The more complicated an algorithm is, the higher the probability of hidden weaknesses.

Adjusting security parameters on existing algorithms avoids the problems inherent in new algorithms. Cryptographic algorithms with easily modifiable security parameters and manageable cost growth (parameter size and computation) are defined here as *Agile Cryptography*. Symmetric cryptography generally has fixed parameter size and is not agile, though some have a few options for key sizes, such as AES, and are somewhat agile. Fortunately, asymmetric algorithms tend to be more agile.

Public key algorithms are based off computationally infeasible problems. Breaking these problems breaks the algorithm. These algorithms are inherently more agile as the size of the hard problem can generally be increased to improve security. In classical public key, there are three commonly used infeasible problems: factoring large integers[16] and computing discrete logarithms over either a multiplicative subgroup of a finite field[4] or an elliptic curve group[10][13].

This gives us the three most common classical foundations for classical public key algorithms (figure 1): factoring, and finite field or elliptic curve based discrete logarithms. Security in each of these algorithms depends on the size of the mathematically hard problem.

- For factoring based algorithms, this is the size and structure of the composite moduli.
- For discrete log based algorithms, it is the size and structure of the cyclic group and the underlying field or elliptic curve.

These can all be easily increased to make existing attacks more difficult – i.e., back to being

computationally infeasible.

Almost 50 years since the start of public key cryptography, a potential new threat has arisen: Shor's algorithm[19] running on a quantum computer. Given a large and stable enough quantum computer, all of these commonly used cryptographically infeasible problems are breakable using Shor's algorithm. The size and stability needed depends on the size and type of problem being attacked. Smaller integers (factoring), smaller base fields (both finite fields and elliptic curves) will need smaller, less stable quantum computers to be attacked than larger integers or fields.

Quantum computing is nowhere near the point of breaking any of these commonly used public key algorithms. Only small, compiled versions of Shor's algorithm for factoring have ever been implemented. Compiled versions of Shor's algorithm use knowledge of the factors to simplify the algorithm, and only 15, 21 and 35 were factored this way. Shor's algorithm for discrete logs, even a very small example, has never been implemented. The life span and security of classical public key algorithms can be easily and cheaply increased by switching to the most agile category, i.e. finite field discrete logarithm based algorithms, thereby increasing security to match evolving threats.

The remainder of this paper details why finite fields are the best classical choice for asymmetric cryptography for both agility and overall security.

2 Classical Comparison

Comparing the agility of factoring, finite field and elliptic curve based cryptography is both simple and nuanced. Parameter simplicity is the primary and simplest reason. To increase security, larger primes for composite moduli (factoring), finite field or elliptic curve bases will be needed. For each category of algorithms, the following will be needed:

- Finite field based algorithms need one large prime, shared by all users. Increasing the size means generating a single new prime for the system.

- Factoring based algorithms require two large primes, p, q for each user of the system and each of these primes must be unique. Increasing the size means securely generating and distributing a new pair of prime integers for every user.
- Elliptic curve based algorithms need a carefully generated curve and new prime base. Generation of a new curve and larger base is only slightly more complicated than its finite field counterpart.

Generating two new primes for each user in a network using factoring based algorithms is far more costly than generating a single prime for the entire network using discrete log based algorithms. In the past, attacks occurred when companies tried to reduce the cost of prime generation and reused primes. This allowed for a very simple attack (see section A.1).

The real issue with elliptic curve agility is its computational growth. For similarly sized base fields, elliptic curve based cryptography is, computationally, many times more expensive (section 4.4) than finite field cryptography. In fact, this has already become an issue when thinking about the threat from quantum computers. Currently sized elliptic curve algorithms are vulnerable[15][17] to a smaller sized quantum computer than their security comparable factoring or discrete log algorithms.

3 Cryptanalytic Sieving Algorithms

Agility in public key algorithms is partially measured by its strength with respect to its size: how much strength is added by increasing parameters? Algorithm strength, in turn, is determined by analyzing known attacks, and the most impactful set of these attacks for discrete logarithm and factoring based systems are the cryptographic sieving algorithms.

Cryptanalytic sieving algorithms are a class of algorithms used to factor integers and compute discrete logs. This set includes quadratic/Gaussian sieve[3], general number field sieve[6], special number field sieve[18], function field sieve[7] and many others. While there are many different varieties, they all share the same three step process:

1. Sieve (section 3.1): Using a chosen number field or function field (think alternative mathematical universe), sieve to generate a huge number of linear equations.

2. Linear Algebra (section 3.3): Given the linear equations generated by the sieve, solve the system.

3. Solve (section 3.2): With the output of the linear algebra step, factor or compute a discrete logarithm.

These steps are superficially the same for factoring and the discrete log, but the devil is in the details. In particular, the linear algebra is acknowledged as the bottleneck for the attacks in record breaking implementations of the discrete log. To deal with this bottleneck, work is pushed off of the linear algebra and onto the sieve and solution steps.

3.1 Sieve

An algorithmic sieve is similar to a physical sieve, separating large particles from small. Primes are the particles and the size of a prime is measured by their norms. This simplified explanation uses positive prime integers whose norm is simply the integer itself[1].

Smooth elements are divisible only by small primes, where small is less than a fixed bound B. For example, if $B = 10$, then 22 would not be smooth (divisible by 11), but 100 would be (the only prime divisors are 2 and 5). Sieves are a very fast way to hunt for these smooth values in their respective number fields.

Once these smooth elements are found, they are converted into a linear equation. Recall that a discrete log base b of an element s is an integer e such that $b^e = s$. If s is a smooth integer with its factorization into primes $s = \prod_{i=0}^{n-1} p_i^{e_i}$, the linear equation resulting from s

[1] Another algebraic structure such as a number field, function field, Gaussian integers, etc., with distinct irreducible/prime elements and a mapping between the problem space (ring of integers mod N or field) gives these sieves their attack strength.

would look like:

$$dlog(s) = \sum_{i=0}^{n-1} e_i dlog(p_i)\,, \tag{1}$$

where *dlog* is a discrete logarithm (and yes, I've ignored the base here).

The cryptanalytic sieves do use two distinct sets of primes $S_1 = \{p_i\}_{i=0}^{n_1-1}$ and $S_2 = \{q_i\}_{i=0}^{n_2-1}$. This time the sieve finds smooth s_1, s_2 with equivalent discrete logs:

$$dlog(s_1) \equiv \sum_{i=0}^{n_1-1} e_{1,i} dlog(p_i) \equiv \sum_{i=0}^{n_2-1} e_{2,i} dlog(q_i) \equiv dlog(s_2) \tag{2}$$

$$0 \equiv \left(\sum_{i=0}^{n_1-1} e_{1,i} dlog(p_i)\right) - \left(\sum_{i=0}^{n_2-1} e_{2,i} dlog(q_i)\right) \tag{3}$$

Sieves search for these equations over a

The system of equations generated by the sieve is then solved using linear algebra and the result used to factor (section 3.2.1) or compute a discrete logarithm (section 3.2.2). Note that the sieving process is massively parallel: the range for the sieve can be divided between many compute engines.

3.1.1 Sieve Costing

The current costing of cryptographic sieving algorithms, and thus the setting of security parameters, is the sieve. Costing for the sieve portion of the attack (which is closely tied to the size of the linear algebra problem to be solved) uses L-Notation. Let n be the integer being factored or the prime modulus for a finite field discrete log problem. Then the cost of the sieving portion for the general and special number field sieve is:

$$L_n\left[\alpha, c\right] = e^{(c+o(1))(\ln n)^{\alpha}(\ln \ln n)^{1-\alpha}}, \tag{4}$$

with

1. n being the integer being attacked (modulus N or prime P)
2. $\alpha = \frac{1}{3}$ and

3. $c = x^{\frac{1}{3}}$ where x depends on the type of sieve (figure 1).

Table 1: Sieving Costs

L-Notation	$L_n\left[a, c\right] = e^{(c+o(1))ln(n)^a ln(ln(n))^{1-a}}$
General **N**umber **F**ield **S**ieve GNFS	$L_n\left[1/3, \left(\frac{64}{9}\right)^{1/3}\right]$
Special **N**umber **F**ield **S**ieve SNFS	$L_n\left[1/3, \left(\frac{32}{9}\right)^{1/3}\right]$

Note that larger bounds on the primes sieved over, i.e., larger sets S_1, S_2, produces a more productive sieve with faster equation generation. Smaller bounds produce fewer equations.

3.2 Solving the Problems

Understanding the last – i.e., solution – step in cryptanalytic sieves is critical to understanding how the factoring/discrete attacks differ. The solution step uses the results from the linear algebra to factor or compute discrete logarithms.

3.2.1 Factoring

One of the common ways to factor a very large integer N is to hunt for two distinct integers a, b modulo N such that:

$$a^2 \equiv b^2 \bmod N$$

or equivalently:

$$a^2 - b^2 \equiv 0 \bmod N$$
$$(a+b)(a-b) \equiv 0 \bmod N.$$

If $a \not\equiv \pm b \bmod N$, then N can't divide either $(a+b)$ or $(a-b)$, and the factors of N must be split between these two values. In other words: if $N = pq$, then p divides $(a+b)$ and q

divides $(a-b)$ or vice versa. The result of taking the **G**reatest **C**ommon **D**ivisor of N and $(a+b)$ or $(a-b)$ produces one of these factors: $\{gcd((a+b),N), gcd((a-b),N)\} = \{p,q\}$.

So for factoring, all we need is to find equivalent squares. The base is irrelevant. All that matters is that the solution ends with all exponents even. In other words, the linear algebra can be done over $\mathbb{F}_2$: one bit per coefficient, addition is an exclusive or ($\oplus$) and multiplication is a logical and ($\otimes$).

3.2.2 Discrete Logarithms

The sieving and linear algebra portions of the attacks compute the *factor base* for an *Index Calculus*[20][12] attack.

The factor base is a collection of computed discrete logarithms for a set of small primes. If we are trying to find the discrete log of b base g – i.e., compute x given b where

$$g^x \equiv b,$$

index calculus attempts to modify b such that the result is smooth. For example, assume the modification of raising b to the 19-th power and multiplying that result by the base g raised to the 101 is smooth:

$$b^{19}g^{101} \equiv \prod p_i^{t_i},$$

where the discrete logs of all p_i ($dlog(p_i)$) are in the factor base. This produces:

$$\begin{aligned} b^{19}g^{101} &\equiv g^{19x+101} \\ &\equiv \prod p_i^{t_i} \\ &\equiv \prod g^{t_i dlog(p_i)}. \end{aligned}$$

Taking the discrete log, base g of both sides and solving gives us x:

$$\begin{aligned} 19x + 101 &\equiv \sum t_i dlog(p_i) \\ x &\equiv \left(\left(\sum t_i dlog(p_i)\right) - 101\right) 19^{-1} \bmod Ord(g) \end{aligned}$$

Note that the larger the factor base of primes, the easier and faster it will be to modify b into a smooth value and the faster this portion of the algorithm will be.

If $Ord(g) = q$, coefficients in the equations from the sieve will be the size of q, generally a large multiple precision integer.

3.3 Linear Algebra

Once the massively parallel sieving process is complete, the equations must be solved. While the sieving step is very similar for both factoring and the discrete log, this step is significantly different.

For discrete logarithms, we need to know the discrete logs modulo the subgroup order – i.e., modulo q where q is a large prime integer. For factoring, all that is needed is equivalent squares (section 3.2.1), so the linear algebra is only concerned with even or odd exponents – i.e., mod 2 or binary arithmetic. For both problems (and as we'll see, particularly for discrete logarithms), keeping the equations sparse is critical to reduce computation cost.

Here's how the storage and computation of coefficients compare for factoring and the discrete log problem with a 1024-bit subgroup using a computer with 64-bit registers:

Table 2: Linear algebra storage and computational differences between factoring and the discrete logarithm problem

What	Factoring	Discrete Log
Storage for single coefficient	$\left(\frac{1}{64}\right)$-th of a register	32 registers (only half the bits can be used for storage for computational purposes)
Addition	$\left(\frac{1}{64}\right)$-th of an xor	32 adds, plus carries and possibly a reduction
Multiplication	$\left(\frac{1}{64}\right)$-th of an AND	32^2 register multiplies, plus carries and reductions; this can be reduced somewhat with special purpose modular multiplication algorithms;

These skewed storage and computation costs, as well as the more difficult problem of

keeping the mod q equations sparse as the reduction progresses, makes the linear algebra portion of the sieving algorithms the real bottleneck for larger discrete logarithms. This is borne out in the records on larger subgroup discrete log implementations: [9][1]. The costing formulas roughly dictate the sieving bounds and the size of the matrix to balance the workload. However, in these very optimized implementations work is shifted off the linear algebra and onto both the sieve and final discrete log computation. Smaller sieving bounds mean fewer variables in the linear algebra but a less productive and more costly sieve. The sieve is also asked to produce more equations per variable to insure a sparser initial matrix.

Computational processes also differ between the sieve and linear algebra. Sieving for equations is generally a massively parallel operation, which can be spread out among as many standard computers as there is access to. The linear algebra, on the other hand is not massively parallel and, particularly for larger problems, requires special purpose machines.

3.4 What the Records Show

Two fairly recent discrete logarithm records allow for comparison between factoring and discrete logarithm sieving attacks. The first, a 2018 discrete logarithm record[9] on a 768-bit prime/767-bit subgroup, compared it to a 2012 factoring record of the same size. The second, a 2020 discrete logarithm record[1] on a 795-bit prime/794-bit subgroup, compared it to their own computation of a 795-bit factoring record.

The write-ups for both records (and others) emphasize the difficulty of the linear algebra and the need to adjust parameters to minimize this cost.

Table 3: Quotes on Linear Algebra Costs

Source	Quote
[8], page 960	For this reason, when using the number field sieve, it is more important to decrease the size of the linear system than when working with the gaussian integer method, even if the system becomes less sparse.
[9], page 8	All that is clear at this point is that attempts to lower this estimate must focus on lowering the linear algebra effort; thus the smoothness parameters must be reduced, but by how much and what the overall effect is going to be is unclear.
[1], page 10	On the other hand for DL, the linear algebra becomes the bottleneck by a large margin if the parameters of the relation search are chosen without considering the size of the matrix they produce.
[1], page 11	In the DL case, we want to limit as much as possible the size of the matrix and the cost of the linear algebra.
[1], page 16	In the DLP-240 case, the choices of the sieving and large prime bounds were dictated by constraints on the size of the resulting matrix.

In both cases, the authors skillfully test and manipulate parameter sizes (sieving numbers up, factor base size down) to minimize the size of the linear algebra to the point where the total core year cost for the discrete logarithm problems is less than four times that of a similarly sized factoring problem. As the authors ([9], page 3) point out, there is no simple direct way of finding the optimal adjustment.

> But more importantly, the very different nature and size of the moduli used in, respectively, the polynomial selection and linear algebra steps imply a radical shift in the trade-off between the steps of the number field sieve, which in turn leads to very different parameter and algorithmic choices compared to what is done for factoring. We are not aware of a satisfactory theoretical analysis of this different trade-off and the resulting parameter selection, or of a reliable way

to predict the practical implication for the relative hardness of integer factoring and prime field discrete logarithm problems. It is clear, however, that the issue is more subtle than recognized in the literature...

The following table displays some of the critical results from the two record discrete logarithm record papers.

Table 4: 768 & 785 bit discrete log records and comparison to factoring

	2012/2018, 768-bit Fact/DL[9]			2020, 795-bit Fact/DL[1]		
	Factoring	Discrete Logs	DL/Fact	Factoring	Discrete Logs	DL/Fact
	in Millions			in Millions		
Raw Relations	64334	10802		8930	3820	
Matrix Size	193	24	$0.124 \approx \frac{1}{8}$	282	36	$0.128 \approx \frac{1}{8}$
Raw Relations per merge variable	333.34	450.08	1.35	31.69	106.23	3.35
Sieving Time (core years)	1500	4000	2.67	794	2400	3.02
Core Seconds per equation	0.735	11.68	15.88	2.80	19.81	7.066
Density (non-zero elements per row)	144	134		200	253	
Density (Percent non-zero per row)	$7.461X10^{-7}$	$1.05X10^{-5}$	14.13	$2.24X10^{-8}$	$6.62X10{-}8$	2.96
Lin. Algebra time (core years)	75	900	12	83	625	7.53
Core Seconds per variable	12.25	1182.6	96.5	9.28	547.5	58.99
Solution (core hours)	20	$200 + 43$	12.15	a	$a \cdot 1000$	~ 1000

Continued on next page

Table 4: 768 & 785 bit discrete log records and comparison to factoring (Continued)

Total Time (core years)	1575.003	4900.03	3.11	877	3026	3.45

Note that while the total core years for the discrete log is less than four times the core years for factoring, there are other comparisons which don't bode well for extending these ratios to larger primes:

- Matrix size for discrete log was reduced to about $\frac{1}{8}$ the size of the factoring matrix;
- More raw relations for each matrix relation were needed for the discrete log problem;
- The sieving time for each raw equation was 15.88 times as long for the 768-bit discrete log and 7.53 times as long for the 795-bit discrete log;
- Linear algebra cost per variable was 96.5 times as long for the 768-bit discrete log and 58.9 times as long for the 795-bit discrete log.
- The larger density differential between the 768 and 795 bit problems is probably partially to blame for the longer run time for the 768 bit problems compared to the 795.
- The costs to reach these optimal parameter settings is not included in the overall cost shown here

4 Myths and False Assumptions

Myths are stories created to simplify more complicated reality. Cryptology has its fair share of myths, and some of them are harmful to security. They start, as most myths do, with some truth, but get warped over time. For example, the true statement "DES key length is too short to be secure against modern computing", is warped to "DES is analytically broken." This myth may dissuade cryptographic designers from using Feistel ciphers – a well studied

and robust foundation for symmetric ciphers. Being aware of these myths may help us make better cryptographic decisions.

4.1 Security of Factoring vs Finite Field Discrete Logarithm

Factoring and finite field discrete logarithm based cryptographic algorithms are often said to have similar strengths for similar modulus size. One of the primary reasons for this comparison is that both algorithms are susceptible to various **C**ryptographic **S**ieving **A**lgorithms (see section 3). These algorithms have three steps:

1. Generate equations (Sieve);
2. Solve equations (Linear Algebra);
3. Solve problem (Solution).

Costing of these algorithms only takes the sieving step into consideration (section 3.1.1). This makes sense for several reasons. First, the costing of the sieve is easy and the costing of the linear algebra is difficult. Second, the sieve is the obvious bottleneck for factoring, which was the focus of most early developments.

For discrete logs, however, this changes. The bottleneck becomes the linear algebra, and it grows more extreme based on both the size of the prime modulus (more variables needed) and the size of the prime subgroup (more memory and computation for each element). Implementers have highlighted this bottleneck by shifting the algorithmic work off of the linear algebra portion and throwing it into the sieve portion.

4.2 Trap Door Primes

Trap door primes are primes which have a special form, enabling the special number field sieve. Recall that the sieve portion of a cryptanalytic sieving attack generates equations by searching over a given area. The sieve portion of the special number field is more productive

– i.e., more equations in a given search area than with the general sieve, lowering the cost (figure 1) . Detecting primes with this form can be difficult, which is why they are called trap door primes.

The largest prime for which a discrete log has been computed to date, a 1024-bit prime[5], was one of these primes. However, this is not the full picture. While the prime for this discrete logarithm was 1024-bits, the subgroup – i.e., the modulus used for the linear algebra – was only 160-bits. For larger subgroups, as noted earlier, the bottleneck for all cryptographic sieve algorithms is the linear algebra.

This is borne out in more recent computed discrete logs. Smaller primes with larger subgroups took much more time[1], even after the work was shifted off of the linear algebra and onto the sieve. Shrinking the factor base bound and increasing the required number of equations from the sieve reduces the number of variables (columns in the resulting matrix) which need to be solved for and improves the sparsity of the resulting matrix. It also makes the sieving far less efficient. The sieve must produce more equations with fewer smooth numbers in their search space.

4.3 One-Size-Fits-All Cryptographically Relevant Quantum Computer

The size and stability needed to make a quantum computer a cryptographically relevant quantum computer depends on the problem the computer is attacking. For cryptography vulnerable to Shor's algorithm, size of the base element is the major factor in determining the number of logical qubits needed. The stability of these logical qubits depends on the number of gates in the implementation – a result of both size and complexity of the problem.

While we currently have no logical qubits and no guarantee of ever obtaining them, one thing we can be fairly sure of is that differing classical public key algorithms (and their related key sizes) require cryptographically relevant quantum computers of different sizes and stability levels. The weakest classical algorithms are those with the smallest base size – elliptic curve cryptography.

4.4 Invulnerability of Elliptic Curve Discrete Logarithm to Cryptanalytic Sieving Algorithm

Cryptographic algorithms based off the discrete logarithm problem work over any finite cyclic group, as long as computing the discrete log is difficult. Discrete log based algorithms were the first[4] to be publicly designed and originally used finite fields.

Elliptic curves were suggested as a base for discrete log algorithms in the late 1980s[10][13]. Point addition in an elliptic curve group replaces multiplication in a finite field, transforming the exponentiation used in finite field cryptography into a scalar multiplication in elliptic curve cryptography. Point addition requires many finite field multiplications. For this reason elliptic curve point addition is much more complicated and costly than a finite field multiply of the same size.

While they are more costly bit-for-bit, elliptic curve groups resist cryptanalytic sieve attacks. This resistance gave designers courage to use very small finite fields, reducing the computational cost to be just a bit more efficient than the larger finite fields. This, combined with the smaller public key size, gave elliptic curves an advantage over finite field based algorithms. Without that small size, elliptic curve cryptography would not be computationally competitive with their finite field counterparts.

Note that resistance to an attack does not imply invulnerability. Elliptic curve groups can be mapped to a finite field[11], generally an extension of the curve's base field, and then attacked with a cryptanalytic sieve. This observation helped stir interest in *function field sieves*: cryptanalytic sieving functions which work best on medium sized prime extension fields[14]. In other words, resistance does not imply invulnerability. To protect against these attacks, curves must be chosen so that all known mappings are into huge extension fields making the attacks less efficient than exhaustion. This puts restrictions on curves and adds the potential of an attack if a new mapping were found.

4.5 Store-now-decrypt-later

Store-now-decrypt-later is one of the primary arguments for moving to Shor resistant algorithms. The reasoning goes like this: attackers can store data secured by classical algorithms and when a cryptographically relevant quantum computer is created that can attack these algorithms, they'll have access to that data.

There are several issues with this argument. First, it assumes a cryptographically relevant quantum computer able to attack these algorithms will exist someday. Second, if a usable cryptographically relevant quantum computer is eventually created, it assumes the data stored now will still have value. But the most critical issue is that it also assumes that the new, less analyzed and tested algorithms will remain secure. I believe that, if anything, store-now-decrypt-later is a stronger argument for shoring up classical algorithms against both classical and quantum attacks until more research on new algorithms is done and more is known about the possibility and timeline for various cryptographically relevant quantum computers.

5 Conclusions

Cryptographic agility is an important feature in secure system designs. Using agile cryptography, algorithms whose parameters can be changed to meet new threats, would be a secure and simplified solution for portions of the cryptographic agility solution. For asymmetric algorithms, discrete logarithms over finite field are the obvious choice. New parameters (primes, generators) are easy to securely generate at larger sizes and to change without disrupting algorithms, particularly compared to both factoring and elliptic curve based algorithms. Discrete logarithm based algorithms are also more secure than factoring based (though more research is needed to determine more accurate security estimates) and for this reason may be a more computationally optimal solution than elliptic curve based algorithms.

A Hard Problems and Background Math

Factoring	Algorithms	RSA, Rabin – Both are mostly used for digital signatures
	Base	Uses modulus $N = pq$ where p, q are large prime numbers, N is public and p, q secret.
Discrete Logarithm	Algorithms	**D**iffie-**H**ellman Key Exchange, ElGamal Signature, **D**igital **S**ignature **A**lgorithm
	Base	**Finite field**: These use a prime integer modulus p and integer g (subgroup *generator*), where g has a large prime order q. In other words, $g^x \not\equiv 1 \bmod p$ all integers $0 < x < q$, but $g^q \equiv 1 \bmod p$. The discrete logarithm problem is as follows: given h, g where $g^s \equiv h \bmod p$, find s: $dlog_g(h) \equiv s \bmod q$.
	Base	**Elliptic curve**: These generally use a prime integer modulus p and an elliptic curve E defined by $$y^2 \equiv x^3 + ax + b \bmod p$$ for specially chosen integers a, b. Curve points are pairs of integers modulo p on the curve: $P_1 = (x_1, y_1)$ with $y_1^2 \equiv x_1^3 + ax_1 + b \bmod p$. Points P_1, P_2 can be added by 1. drawing a line through the curve at P_1, P_2; 2. finding the third point, $P_3 = (x_3, y_3)$, intersecting the curve and line, 3. then taking its inverse: $-P_3 = (x_3, -y_3)$. So $P_1 + P_2 = (-P_3)$. As with finite fields, a generator – in this case a point P with a large prime order q – where $\overbrace{P + P + \ldots + P}^{q} = qP = 1$. The discrete logarithm problem is as follows: given P, Q where $sP = Q$, find s: $dlog_P(Q) \equiv s \bmod q$

Figure 1: Classical Computationally Infeasible Problems and Derived Algorithms

A.1 Factoring with GCDs

If factoring is the hard problem used in a public key algorithm, then every user must have their own set of distinct primes. Two users cannot share a modulus or they will not have unique keys and will be able to break each others systems. Even sharing one prime divisor of their modulus leads to a trivial attack.

Let $N_1 = p_1 q$ and $N_2 = p_2 q$. These moduli share a common divisor: q. Computing a **G**reatest **C**ommon **D**ivisor of N_1, N_2 would give an attacker $gcd(N_1, N_2) = q$, and the complete factorization of both N_1 and N_2.

References

[1] F. Boudot, P. Gaudry, A. Guillevic, N. Heninger, E. Thomé, and P. Zimmermann, *Comparing the difficulty of factorization and discrete logarithm: a 240-digit experiment*, 2020, Cryptology ePrint Archive, Report 2020/697, `https://eprint.iacr.org/2020/697`.

[2] Zhengjun Cao and Zhenfu Cao, *The systemic errors of banded quantum fourier transformation*, Cryptology ePrint Archive, Paper 2024/454, 2024, `https://eprint.iacr.org/2024/454`.

[3] R. Crandall and C. Pomerance, *Prime numbers: A computational perspective*, second ed., ch. 6, pp. 227–244, Springer-Verlag, 175 Fifth Avenue, New York, New York 10010, U.S.A., 2005.

[4] Whit Diffie and Martin Hellman, *New directions in cryptography*, Transactions on Information Theory, no. 22, IEEE, November 1976, pp. 644–654.

[5] Joshua Fried, Pierrick Gaudry, Nadia Heninger, and Emmanuel Thomé, *A kilobit hidden* SNFS DISCRETE LOGARITHM COMPUTATION, July 2016, `https://eprint.iacr.org/2016/961.pdf`.

[6] D.M. Gordon, *Discrete logarithms in $gf(p)$ using the number field sieve*, SIAM Journal on Discrete Mathematics (1993), no. 6, 124–138.

[7] Antoine Joux, *Faster index calculus for the medium prime case application to 1175-bit and 1425-bit finite fields*, Advances in Cryptology – CRYPTO **7881** (2013), 177–193.

[8] Antoine Joux and Reynald Lercier, *Improvements to the general number field sieve for discrete logarithms in prime fields. a comparison with the gaussian integer method*, Mathematics of Computation **72** (2002), no. 242, 953–967, https://www.ams.org/mcom/2003-72-242/S0025-5718-02-01482-5/S0025-5718-02-01482-5.pdf.

[9] Thorsten Kleinjung, Claus Diem, Arjen K. Lenstra, Christine Priplata, and Colin Stahlke, *Computation of a 768-bit prime field discrete logarithm*, 2017, https://eprint.iacr.org/2017/067.pdf.

[10] Neal Koblitz, *Elliptic curve cryptosystems*, Mathematics of Computation **48** (1987), no. 177, 203–209, https://www.ams.org/journals/mcom/1987-48-177/S0025-5718-1987-0866109-5/S0025-5718-1987-0866109-5.pdf.

[11] Alfred J. Menezes, Tatsuaki Okamoto, and Scott A. Vanstone, *Reducing elliptic curve logarithms to logarithms in a finite field*, IEEE Trans. Inform. Theory **39** (1993), no. 5, 1639–1646.

[12] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, *Handbook of applied cryptography*, ch. 3.6.5, CRC Press, 1996.

[13] Victor Miller, *Use of elliptic curves in cryptography*, Advances in Cryptology – CRYPTO **85** (1986), 417–426, https://link.springer.com/chapter/10.1007/3-540-39799-X_31.

[14] Madhurima Mukhopadhyay, Palash Sarkar, Shashank Singh, and Emmanuel Thomé, *New discrete logarithm computation for the medium prime case us-*

ing the function field sieve, Advances in Mathematics of Communications **0** (2020), no. 1930-5346 2019 0 109, NA, http://aimsciences.org//article/id/d7f51db9-e046-4282-82a7-8c1217b53808.

[15] John Proos and Christof Zalka, *Shor's discrete logarithm quantum algorithm for elliptic curves*, 2004, https://arxiv.org/pdf/quant-ph/0301141.

[16] R.L. Rivest, A. Shamir, and L.M. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM (1978), no. 21, 120–126.

[17] Martin Roetteler, Michael Naehrig, Krysta M. Svore, and Kristin Lauter, *Quantum resource estimates for computing elliptic curve discrete logarithms*, Lecture Notes in Computer Science (2017), 241–270, https://arxiv.org/abs/1706.06752.

[18] I.A. Semaev, *Special prime numbers and discrete logs in finite prime fields*, Mathematics of Computation **71** (2002), no. 237, 363–377, https://www.ams.org/journals/mcom/2002-71-237/S0025-5718-00-01308-9/S0025-5718-00-01308-9.pdf.

[19] Peter W. Shor, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM Journal on Computing **26** (1997), no. 5, 1484 – 1509.

[20] A.E. Western and J.C.P. Miller, *Tables of indices and primitive roots*, Royal Society mathematical tables, Published for the Royal Society at the University Press, 1968.