Security levels in cryptography

Steven Galbraith

University of Auckland, New Zealand

Thanks

- Thanks to Joseph K. Liu and Hui Cui (Program Chairs) for inviting me to give the Jennifer Seberry Invited Lecture.
- Thanks to everyone else involved with running ACISP this year.
- Thanks Ben Smith and Samuel Dobson, for a research collaboration that started me thinking about this problem.
- Thanks to Alfred Menezes, Mihir Bellare, Kenny Paterson and Martin Albrecht for comments and suggestions and references.

Plan

- Discuss literature on the meaning of “security level” and give a definition.
- Discuss Pollard rho and the security level of the elliptic curve discrete logarithm problem.
- Tightness (if time).
- Multi-user security (no time).
- Groups of unknown order (if time).

Security definitions in theoretical cryptography

- The established definition of concrete single-key security (with security parameter λ) is: Every adversary A that runs in time $t = poly(\lambda)$, succeeds with negligible probability $\epsilon(\lambda)$.
- The concrete security concept goes back to work of Bellare and Rogaway and their co-authors in the mid-90s.
- One of the main topics of this talk is: What values for t and ϵ are of interest?

Brute-force attack on symmetric encryption

- Let Enc and Dec be a symmetric cipher with λ-bit keys.
- Let $k \in \{0,1\}^\lambda$ be sampled uniformly at random.
- Consider an adversary that is given one or more pairs $(m_i, \mathrm{Enc}_k(m_i))$ and wishes to determine the key k.
- The adversary can do the **brute-force attack** of trying keys and encrypting m_1 (or decrypting c_1). (Additional messages might be needed if more than one key encrypts m_1 to c_1.)
- The worst case running time of the attack is 2^λ executions of the encryption/decryption function. The average case running time is $2^{\lambda-1}$ executions.
- The attack requires low storage and can be parallelised.

Security of symmetric encryption

- If there is no known attack on a symmetric cipher with λ-bit keys that is better that brute-force, then we say the cipher has λ bits of security.
- For example, the wikipedia page for "Security Level" says "*In cryptography, security level is a measure of the strength that a cryptographic primitive - such as a cipher or hash function - achieves. Security level is usually expressed in bits, where n-bit security means that the attacker would have to perform 2^n operations to break it*".

How much computation can we do?

- Twitter tells me that the total bitcoin mining rigs today computes around $2^{66.7}$ hashes per second.
- This is approximately $2^{91.6}$ hashes per year. Or 2^{100} hashes in 338 years (ignoring Moore's law).
- These rigs are customised to compute hashes quickly, and are not suited to other cryptanalysis operations.
- Hence, computations of more than 2^{100} operations seem very remote at present.
- Current systems (e.g., AES) are expected to have at least 128 bits of security.

Success probabilities

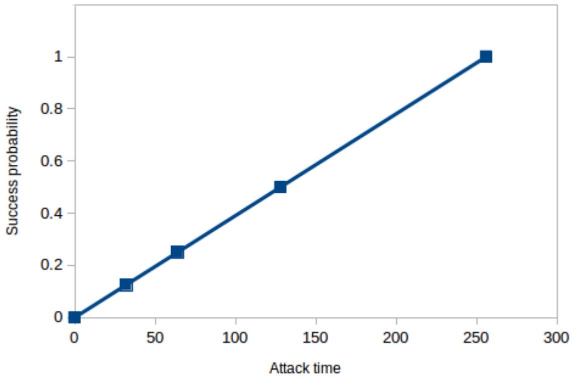
- In algorithmic number theory one considers (randomised) algorithms for computational problems that succeed with probability close to 1.
- In a computational number theory software package, an algorithm that succeeds with probability $1/2^{10}$ is useless – the customer will give up and use a different package.
- In contrast, in crypto, an attacker that breaks the scheme with probability $1/2^{10}$ is a killer attack and the system is dead.

Security of symmetric encryption

- Let us consider again a symmetric cipher with λ-bit keys.
- One can guess k bits of the secret key and try all the $2^{\lambda-k}$ keys with that pattern. The resulting attack requires computing at worst $t = 2^{\lambda-k}$ encryptions and succeeds with probability $\epsilon = 1/2^k$.
- Note that $t/\epsilon = 2^\lambda$.
- Alternatively, $t = \epsilon 2^\lambda$.

Running time of brute force search over key size 2^λ

This is the graph of $t = \epsilon 2^\lambda$.



Formal definition of success probability

- When defining security of a system there is a key/instance generation algorithm Gen that outputs instances x.
- The attacker is a randomised algorithm A.
- The success probability of the attacker for given security level λ is the probability that $A(x)$ returns a correct solution to the instance, where the probability is over $x \to \mathrm{Gen}(1^{\lambda})$ and over the random choices made by A.

Definition of security level

- **Definition:** Let X be a computational problem, with instances x produced by an algorithm $\mathrm{Gen}(1^\lambda)$. Then X has λ-bit **security level** if, for every adversary A, if $A(x)$ runs in time t and succeeds with probability ϵ we have $t/\epsilon \geq 2^\lambda$.
- I consider this definition as folklore.
- Lenstra and Verheul (Selecting cryptographic key sizes, 2001) use the phrase "incomplete attacks" to mean attacks with success probability less than one.
- The model in their paper is to define security as equivalent to breaking a symmetric cipher, so the relation $t/\epsilon \geq 2^\lambda$ is implicit in their paper.

Definition of security level

This definition appears in several papers, including:

- Mihir Bellare and Thomas Ristenpart, Simulation without the Artificial Abort: Simplified Proof and Improved Concrete Security for Waters' IBE Scheme, 2009.
  The ratio t/ϵ is called the "work factor" in Section 4.
- Daniele Micciancio and Michael Walter, On the Bit Security of Cryptographic Primitives, 2019.
  They use this definition for search problems, and also discuss security levels for decision problems.
- Mihir Bellare and Wei Dai, The Multi-Base Discrete Logarithm Problem: Concrete Security Improvements for Schnorr Identification, 2020.

Cost metrics

- One should take storage costs and memory access into account when analysing algorithms.
- In 2004, Wiener ("The full cost of cryptanalytic attacks") addressed this.
- Similarly, Bernstein and Lange discusses the "AT metric" (meaning the product of the area of a chip and the running time) in their 2013 paper "Non-uniform cracks in the concrete".
- Also in almost all cryptanalysis settings one can make use of parallelism, so it should be understood that the cost estimate is for the total computation.

Types of attack with success < 1

There are two ways that an algorithm could succeed with probability < 1:

- It might only work for some subset of “easy instances” or “weak keys”.
  - The Fermat compositeness/primality test fails on Carmichael numbers, but does correctly determine that a large class of composite integers are not primes.
  - The LLL algorithm correctly computes the shortest vector in a lattice in certain cases (e.g., when the shortest vector is much shorter than the second successive minimum).
- The algorithm may make unlucky choices.
  - The Miller-Rabin primality test (choosing a fixed number of random bases) may incorrectly declares some composite numbers to be prime.

Amplifying success

- For many (but not all!) problems, an algorithm that succeeds with probability $\epsilon < 1$ can be transformed into an algorithm that succeeds with probability close to 1.
- This can happen because it is a randomised algorithm and executions are independent experiments.
    - The "guess *k* bits" algorithm for key search in a symmetric cipher can be repeated with a different guess.
    - Repeating the Miller-Rabin primality/compositeness testing algorithm with an independent set of random bases increases the probability the result is correct.
- Another class of algorithms can increase their success using random self-reductions.

Random self-reductions

- A random self-reduction takes an instance X of a computational problem and outputs an instance X' that is independent of X, such that a solution to the instance X' can be transformed into a solution to the instance X.
- For example, suppose P, Q is an instance of the elliptic curve discrete logarithm problem, so that $Q = [a]P$ for some secret integer a.
- Let r be the order of P (assumed to be prime).
- Choose uniformly $1 \leq u < r$ and $0 \leq v < r$ and set $P' = [u]P$ and $Q' = Q + [v]P$.
- Note that P' also has order r and that P' and Q' are uniformly distributed in the (sub)group and independent of (P, Q).
- If $Q = [a']P$ then $a = a'u - v \pmod{r}$.

Amplifying success

- Suppose that algorithm A succeeds with probability ϵ and that executions of A are independent.
- Then the probability A does not succeed after k trials is $(1-\epsilon)^k$.
- Approximating $1-\epsilon \approx e^{-\epsilon}$ we have $(1-\epsilon)^k \approx e^{-k\epsilon}$.
- Taking $k = 1/\epsilon$ means A fails with probability $\approx e^{-1} \approx 0.36$ and so succeeds with probability $\approx 0.63 > 0.5$.
- Hence, for problems that have random self-reductions, there can't be a noticeable proportion of weak instances without weakening the general case.

The converse of amplifying success

- Random self-reductions can transform an algorithm that succeeds with probability ϵ into an algorithm that succeeds with probability close to 1.
- **The converse of this argument doesn't automatically hold!**
- The existence of an algorithm that solves a problem in time t with probability 1 does not imply the existence of an algorithm that succeeds with probability ϵ and solves the problem in time $t\epsilon$.
- Studying algorithms with success probability ≈ 1 does not shed light on algorithms with success probability $\ll 1$.

Small success probabilities

- The literature mainly focusses on the case $\epsilon = 1$ and does not consider attacks with success probability $\epsilon < 1$.
- For example, the Handbook of Applied Cryptography by Menezes, Van Oorschot and Vanstone, only considers success probability 1. They define a notion called the "work factor":

  1.69 Definition: *The work factor W_d is the minimum amount of work (measured in appropriate units such as elementary operations or clock cycles) required to compute the private key d given the public key e, or, in the case of symmetric-key schemes, to determine the secret key k.*

- In other words, security levels seem to be usually calculated based on the case of success probability 1.

Why 128-bits? (My personal opinion)

Using 128-bit keys is a conservative choice to protect against attackers who succeed with small probability ϵ and run in time $t = \epsilon 2^{128}$.

We are not actually worried about attackers that perform 2^{128} operations.

Section 3.1 of Katz and Lindell

One might want to use a scheme with the guarantee that no adversary running for at most 200 years using the fastest available supercomputer can succeed in breaking the scheme with probability better than 10^{-30}*. It is instructive to get a feel for values* t *and* ϵ *that are typical of modern cryptographic schemes.*

Example 3.1 *Modern private-key encryption schemes are generally assumed to give almost optimal security in the following sense: when the key has length* λ*, an adversary running in time* t *(measured in, say, computer cycles) can succeed in breaking the scheme with probability at most* $t/2^{\lambda}$*.*

A prudent choice of parameters would be $t = 2^{80}$ *and* $\epsilon = 2^{-48}$ *(implying that the key must be at least 128 bits long).*

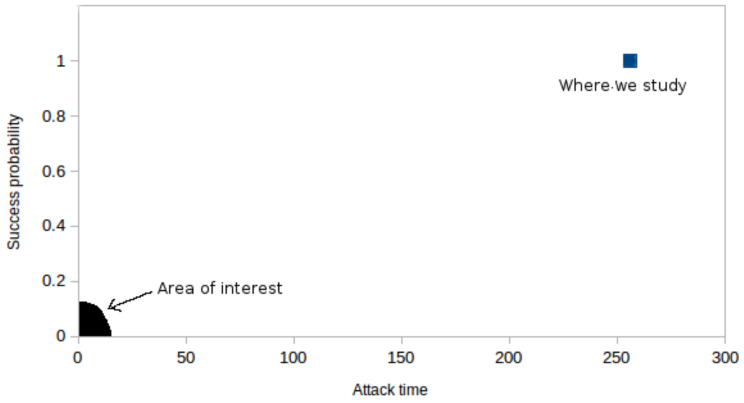
Bellare and Dai, INDOCRYPT 2020

- Bellare and Dai (The Multi-Base Discrete Logarithm Problem: Tight Reductions and Non-Rewinding Proofs for Schnorr Identification and Signatures) also discusses these issues.
- Their paper contains, in various places, the pairs $(t, \epsilon) = (2^{90}, 2^{-32}), (2^{80}, 2^{-48}), (2^{64}, 2^{-64})$.
- They never worry about the case $(t, \epsilon) = (2^{128}, 1)$.
- They talk of the "full curve of advantage as a function of runtime".
- For 256-bit security they consider $(t, \epsilon) = (2^{100}, 2^{-156})$.

Small success probabilities

- The main theme of this talk is that to evaluate concrete security of some computational problem X needs a careful analysis of algorithms with success probability < 1.
- I argue it is *never* interesting to consider $\epsilon = 1$.
- **Revised Definition of Security Level:** Let X be a computational problem, with instances x produced by an algorithm $\mathsf{Gen}(1^\lambda)$.
  Let ϵ_0 be some fixed upper bound on success probabilities of interest (possibly a function of λ).
  Then X has λ-bit **security level** if, for every adversary A, if $A(x)$ runs in time t and succeeds with probability $\epsilon < \epsilon_0$ we have $t/\epsilon \geq 2^\lambda$.
- For key search the revised definition is equivalent to the earlier one.

Small success probabilities



Elliptic curve discrete logarithm problem (ECDLP)

- Let E be an elliptic curve over a finite field $\mathbb{F}_q$.
- Let $P \in E(\mathbb{F}_q)$ be a point of prime order r.
- The ECDLP is: Given $Q \in \langle P \rangle \subseteq E(\mathbb{F}_q)$ to compute $a \in \mathbb{Z}_r$ such that $Q = [a]P$.

Pollard rho

- The baby-step-giant-step algorithm requires large storage, is hard to parallelise or distribute, and costs more than square-root in realistic cost models.
- The rho and kangaroo algorithms require less storage and can be distributed.
- They are based on pseudorandom walks, and compute group elements $R_i = [u_i]P + [v_i]Q$.
- A collision $R_i = R_j$ means $[u_i]P + [v_i]Q = [u_j]P + [v_j]Q$ and so

$$Q = [(u_i - u_j)(v_j - v_i)^{-1} \pmod{r}]P.$$

Pollard rho success probability

- The success probability of rho is analysed based on the birthday paradox.
- We need the probability of a collision when sampling t elements uniformly at random with replacement from a set of size r.
- The probability the elements are all distinct (and hence, no collision) is

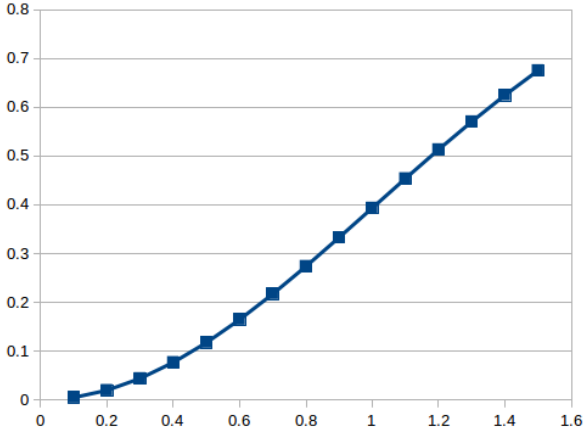
$$1(1-\tfrac{1}{r})(1-\tfrac{2}{r})\cdots(1-\tfrac{t-1}{r}) \approx \exp(-t^2/2r).$$

- Hence the probability of success for Pollard rho after sampling t group elements is approximately $1-\exp(-t^2/2r)$, which is low when $t \ll \sqrt{r}$.

Pollard rho success probability

- Precisely, when $t \ll \sqrt{r}$ the probability to solve the DLP using Pollard rho satisfies $\epsilon < t/\sqrt{r}$.
- Alternatively $t/\epsilon > \sqrt{r}$.
- So Pollard rho, like baby-step-giant-step, does not satisfy the linear relationship $t/\epsilon = 2^\lambda$ of brute-force key search.
- This phenomenon was already noted by Lenstra and Verheul.

Pollard rho success probability



Pollard rho success probability

- We now determine the group order such that the success probability of Pollard rho running in time $t = 2^{80}$ is approx $\epsilon = 2^{-48}$.
- Solve $\epsilon = 1 - \exp(-\theta^2/2)$, which for $\epsilon = 2^{-48}$ gives $\theta = 8.4 \cdot 10^{-8}$.
- Then $t = \theta\sqrt{r}$ so $r = (2^{80}/\theta)^2 \approx 2^{206}$.
- Hence a 206-bit group order suffices if the attacker is using Pollard rho and can not do more than 2^{80} group operations.

Controversial opinion

We are overly conservative in our group sizes for 128-bit security, and 206-bits is enough for 128-bit secure ECC.

Equivalent to AES

- It has become common in public key crypto to talk about security levels "equivalent" to AES with some given key size.
- For example, the NIST post-quantum crypto requires submitters to give parameters for various levels.
- NIST level one: *Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for key search on a block cipher with a 128-bit key (e.g. AES128).*
- But this comparison is undefined, as symmetric crypto has a linear (t, ϵ) curve while public key assumptions like ECDLP, lattices, isogenies, etc do not.

Remainder of talk

One can study many problems through this lens:

- Tightness.
- Multi-user security (various versions).
- Post-quantum assumptions.
- Groups of unknown order.

But I am short on time!

Tightness

- Recall that a security reduction from a cryptosystem to a computational problem shows that if an attacker A breaks the cryptosystem (in some model) when running in time t and with success probability ϵ, then there is an algorithm A' to solve the computational problem that runs in time t' and has success probability ϵ'.
- Here t' and ϵ' are functions of t, ϵ and the security parameter λ.
- A reduction is called *tight* if $t' \approx t$ and $\epsilon' \approx \epsilon$.
- So a tight reduction satisfies $t'/\epsilon' \approx t/\epsilon$.

Tightness

- A reduction is *loose* if $t'/\epsilon' \gg t/\epsilon$, which can happen if t' is much bigger than t (the algorithm A' is very slow) or if ϵ' is much smaller than ϵ (the algorithm A' succeeds with much lower probability).
- Loose reductions are not helpful to set practical security levels: if one wants $t/\epsilon > 2^\lambda$ then one has t'/ϵ' much larger and so, in principle, needs to choose parameters so that the computational problem has security level t'/ϵ'.

Tightness

- This issue is discussed by Koblitz and Menezes in the first of their "another look" papers.
- Section 5.5 discusses security proofs for discrete-log-based signatures using the Pointcheval-Stern Forking Lemma.
- Consider a forger that runs in time t, makes at most q_H queries to the random oracle, and succeeds with probability ϵ.
- Using the generalised forking lemma of Bellare and Neven one gets $t' \approx 2t$ and

$$\epsilon' \approx \epsilon^2 / q_H.$$

Tightness

- What is interesting, from our point of view, is that Koblitz and Menezes immediately move to the situation of success probability 1.
- They write "we have to run the forger program k times in order to find (with high probability) the discrete logarithm".
- They conclude that 80-bit security requires a 354-bit order, rather than 160 bits.
- (Admittedly, Koblitz and Menezes focus on an attacker that has success probability $\epsilon \approx 1$ and are mostly concerned with the q_H factor, which they take to be the most extreme value $q_H = t$.)

Tightness

- What happens if we do not amplify to success probability 1?
- Following Katz and Lindell we might take $t = 2^{80}$, $\epsilon = 2^{-48}$ and $q_H = 2^{40}$.
- Then $\epsilon' \approx \epsilon^2 / q_H = 2^{-136}$ which is smaller than $1/2^{\lambda}$.
- Such a small probability may seem irrelevant, since one can guess a λ-but key with probability $1/2^{\lambda}$.
- But for number-theoretic systems, attacks with such a low probability may not be absurd.
- One can compute the minimum group size r such that Pollard rho runs in time $t = 2^{80}$ and succeeds with probability 2^{-136}:
  To have $1 - \exp(-t^2/2r) = 2^{-136}$ can take $r \approx 2^{296}$.
- So 296-bit ECC may be tight for digital signatures.

Tightness

- Maybe 2^{-136} is far too small to be worth considering.
- Suppose we repeat the attack k times to get an algorithm with success probability $1 - (1 - \epsilon')^k \approx k\epsilon'$.
- Taking $k = 2^{136}$ to get success probability 1 is the situation considered by Koblitz and Menezes,
- Aiming for $k\epsilon' = 2^{-48}$ one would take $k = 2^{88}$, giving an algorithm in time $t = 2^{168}$.
- What group size is required for Pollard rho to run in time $t = 2^{168}$ and succeed with probability 2^{-48}?
- Answer: $r \approx t^2/\epsilon = 2^{384}$, which is not much bigger than the 354 bits for only 80-bit security calculated by Koblitz and Menezes.

Bellare and Dai, INDOCRYPT 2020

- Bellare and Dai ("The Multi-Base Discrete Logarithm Problem: Tight Reductions and Non-Rewinding Proofs for Schnorr Identification and Signatures") have given similar analysis about group sizes for tight reductions.
- They also give a new way to prove security of Schnorr signatures, based on a new (interactive) computational assumption.

Multi-user security

- Consider a cryptosystem with n users.
- We might be interested in the cost to attack one random user out of the n.
- Or might want to break the system for *all* n users.
- Ideally a system would be such that there is no faster way to solve all n instances.
- This is measured in the **scaling factor** of Auerbach, Giacon and Kiltz (Everybodys a Target: Scalability in Public-Key Encryption).

Using a FIXED Group

- In practice we use fixed groups, such as Curve25519.
- Bernstein and Lange's paper "Non-uniform cracks in the concrete" presents precomputation approaches, expressed as an exponentially large hint.
- They argue that in the AT metric there is no speedup of such methods.
- There is related/following work by several authors, including Lee, Cheon and Hong; Corrigan-Gibbs and Kogan.
- All these works require a precomputation that exceeds 2^{128} operations. Hence I argue the results are irrelevant in practice.

Groups of unknown order

- An active area of research is **groups with unknown order**: no entity in the system knows the order of the group.
- These have been used for accumulators since Benaloh-de Mare (1993) and delay functions/time lock puzzles since Rivest-Shamir-Wagner (1996).
- For accumulators a computational problem of interest is the *low order assumption*.
- In the low-order assumption an attacker is given a description of a group G, and is required to produce an element $g \in G$ of order d, where $1 < d < 2^\lambda$.
- This assumption makes no sense for a fixed group: attacker with a hard-wired group element g exists and is efficient.

Sutherland's algorithm

- Suppose one has an abelian group G of order bounded by some integer N, and wants to compute the order.
- One could sample a random element g and use Pollard rho to compute the order of g in $O(\sqrt{N})$ group operations.
- Sutherland's idea is to first raise g to a large smooth power E, in the hope that the order of $g' = g^E$ is bounded by $M \ll N$.
- If so, one can compute the order of g' in $O(\sqrt{M})$ group operations, and then work back to the order of g.
- Fix some $u > 2$. Suppose that the order of G can be written as $n_1 n_2$ where n_1 is $N^{1/u}$-smooth (meaning all prime powers dividing n_1 are smaller than $N^{1/u}$), and $n_2 \leq N^{2/u}$.
- Then Sutherland shows that one can compute the order of G in time $\tilde{O}(N^{1/u})$ group operations.

Sutherland's algorithm

- Sutherland's algorithm is not effective if the smooth part of the order of G is not large enough, which means the remaining computation using Pollard rho is too slow.
- One can estimate the probability that a random integer in $[1, N]$ can be written as $n_1 n_2$ where n_1 is $N^{1/u}$-smooth and $n_2 \leq N^{2/u}$.
- For example, taking $u = 6$ gives probability 0.00109.
- If the orders of groups G produced by a random group generator behave like random integers in $[1, N]$, then Sutherland's algorithm can be effective at counting points for some such groups.
- For example, there is an algorithm that runs in time $N^{1/6}$ group operations and succeeds with probability $\epsilon = 0.00109$.

Smoothness probabilities

Asymptotic probability a random integer n in $[1, N]$ can be written as $n_1 n_2$ where n_1 is $N^{1/u}$-smooth and $n_2 \leq N^{2/u}$.

u	prob	u	prob	u	prob
2.1	0.9488	5.0	0.4473	12.0	4.255e-12
2.9	0.5038	6.0	1.092e-03	16.0	6.534e-19
3.0	0.4473	10.0	5.382e-09	20.0	2.416e-26

Class groups as unknown order groups

- It has been suggested that approx 2000-bit negative fundamental discriminant (which means an approximately 1000-bit group order) should provide 128-bit security.
- Sutherland's algorithm with $u = 6$ would require $(2^{1000})^{1/6} > 2^{166}$ operations and so would not be a concern.
- But taking $u = 12$ gives an algorithm performing (proportional to) 2^{84} operations and succeeding with probability $\epsilon = 4.255 \cdot 10^{-12} \approx 2^{-38}$.
- Taking $t = 2^{84}$ and $\epsilon = 2^{-38}$ we have security level $t/\epsilon = 2^{122}$.
- But is this the right calculation?

Using fixed groups of unknown order

- Applications such as accumulators require a fixed group that is shared by many users.
- One cannot meaningfully formulate the point-counting problem or the low order assumption for a fixed group.
- So our previous discussion of security levels does not apply in this setting.

Using fixed groups of unknown order

- Recall the low-order assumption: an attacker is given a description of a group G, and is required to produce an element $g \in G$ of order d, where $1 < d < 2^\lambda$.
- There is no random self-reduction, so there is no way to amplify the success as we did for ECDLP or other problems.
- Instead, we have a class of "weak instances": outputs of the group generating algorithm whose group order makes them susceptible to Sutherland's algorithm.
- There is no way to efficiently identify a weak instance.
- If the instance is not from a weak class, then there is no particular speedup to the known algorithms (i.e., no analogue of algorithms of the form "guess some bits and compute the rest").

Using fixed groups of unknown order

- For applications such as accumulators that use a fixed group of unknown order, a new definition of security level is needed (work in progress with Dobson and Smith).
- The probability that the fixed group is vulnerable to Sutherland's algorithm needs to be very small.
- Going back to Katz and Lindell, we might want the probability the group is weak to be $\epsilon = 2^{-48}$.
- Such issues do not arise if a fresh group is generated for each execution of the protocol (as is done in the Chia blockchain, for example).

Controversial opinion

When using a fixed group of unknown order, we should take the group order to be at least 1000 bits to minimise the risk that Sutherland's algorithm is the best attack.

This may require a redesign of schemes currently using class groups (but not Chia).

Controversial opinion

It's better to use divisor class groups of genus 3 curves than ideal class groups of quadratic fields (see Dobson, Galbraith and Smith eprint 2020/196).

Suggested parameters for fixed groups of unknown order

- Cautious: 1900 bits
  With probability $1 - 2^{-55}$ there is no algorithm to compute the order with running time less than 2^{128}.
- Paranoid: 3320 bits
  With probability $1 - 2^{-128}$ there is no algorithm to compute the order with running time less than 2^{128}.

Conclusion

- I have presented a (folklore) definition of “security level”.
- I argue that one could consider smaller key sizes for elliptic curve crypto.
- I claim the notion “equivalent to AES” is not well-defined for many number-theoretic problems.
- It seems that non-tight security proofs may not require as big an increase in group sizes as previously thought.
- I argue that traditional notions of security level are not suitable for groups of unknown order.

Controversial opinion

Algorithmic number theorists should pay more attention to algorithms with small success probability, especially in lattice and isogeny crypto.

Thank You