On the Security of 1024-bit RSA and 160-bit Elliptic Curve Cryptography

version 2.1, September 1, 2009

Joppe W. Bos[1], Marcelo E. Kaihara[1], Thorsten Kleinjung[1], Arjen K. Lenstra[1,2], and Peter L. Montgomery[3]

[1] EPFL IC LACAL, Station 14, CH-1015 Lausanne, Switzerland
[2] Alcatel-Lucent Bell Laboratories
[3] Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA

Abstract. Meeting the requirements of NIST's new cryptographic standards means phasing out usage of 1024-bit RSA and 160-bit elliptic curve cryptography (ECC) by the end of the year 2010. This write-up comments on the vulnerability of these systems to an open community attack effort and aims to assess the risk of their unavoidable continued usage beyond 2010 until the migration to the new standards has been completed. We conclude that for 1024-bit RSA the risk is small at least until the year 2014, and that 160-bit ECC over a prime field may safely be used for much longer – with the current state of the art in cryptanalysis we would be surprised if a public effort can make a dent in 160-bit prime field ECC by the year 2020. Our assessment is based on the latest practical data of large scale integer factorization and elliptic curve discrete logarithm computation efforts.

Keywords: Key management, NIST Special Publication 800-57, Suite B Cryptography, 80-bit security, RSA, integer factorization, NFS, ECC, Elliptic curve discrete logarithm, Pollard rho

1 Introduction

In February of 2005, the United States' National Security Agency (NSA), arguing that *"The sustained and rapid advance of information technology in the 21st century dictates the adoption of a flexible and adaptable cryptographic strategy for protecting national security information"*, announced its *Suite B of Cryptographic Protocols* (cf. [36]). Along similar lines, the United States' National Institute of Standards and Technology (NIST) provides guidance *to use algorithms that adequately protect sensitive information, and to plan ahead for possible changes in the use of cryptography* in *Special Publication 800-57, Part 1* (cf. [34]; see also [35] and [32]). These initiatives are targeted broadly enough that the majority of vendors and major enterprises – worldwide, not just in the USA – will be more or less forced to follow them.

Roughly speaking, "Suite B" adopts 128-bit and higher security levels, leaving behind the current widely-employed 80-bit security level. It also prescribes a precise set of technologies to be used. NIST's "SP 800-57 (1)" allows more flexibility in the technology used and settles for 112-bit security at the lowest level from the year 2011 on. Any migration – to other technologies, to a new security level, or to both – will be a logistical nightmare and has the potential for substantial security breaches unrelated to the cryptographic security levels.

Nevertheless, migration to a security level higher than 80 bits is not just mandatory, there is also no question that it is a very good idea. The migration should, however, be orchestrated with utmost care and too hurried adoption of the new standards should be avoided; even for the NSA itself it will take many years to comply (cf. [17]) and NIST, in [35], comments that *We've learned that this is not so easily done.* Thus, though both "Suite B" and "SP 800-57 (1)" are excellent initiatives whose implementation by the year 2010 can be recommended, an important question is how much time companies and other complying practitioners can reasonably allow themselves before fully adopting either of the new standards, in a manner that is as cost effective and security conscious as possible. This latter question is addressed in this paper. It should not be construed as support for the old 80-bit security level, but points out some of the realistic risks of completing migration to a higher security level at a date beyond 2010.

The NSA and NIST initiatives affect all three pillars of modern cryptography: block ciphers, cryptographic hash functions, and public key cryptography. For block ciphers "Suite B" requires the Advanced Encryption Standard (AES), which was designed to provide 128-bit security or more. Although there are doubts about the adversarial model in which the intended security level of AES is reached, there is consensus that single DES (the classical Data Encryption Standard) can no longer be recommended even for low risk applications. "SP 800-57 (1)" also allows limited usage of three key triple DES. This write-up does not further deal with the subject of block ciphers or migration to AES. It does not deal with cryptographic hash functions either. But it may be good to remind the reader that the situation with respect to cryptographic hash functions is disconcerting. In the first place, there are concerns about the security provided by the current standard, the 160-bit hash function SHA-1 which is designed to provide 80-bit security. Convincing arguments have been presented that the security level of SHA-1 is quite a bit lower than that: right now it is not clear if the bottom is in sight yet. If chosen-prefix collisions as presented in [42] for MD5 could be made to work for SHA-1, attacks may be feasible that could have an undesirably negative impact on the security of the Internet as a whole. SHA-256, as required by "Suite B", may be regarded as an improvement over the common use of SHA-1. However, given the design similarities between SHA-1 and SHA-256 and the currently unforeseeable outcome of the NIST hash competition, the situation with respect to cryptographic hashing is unclear.

Here we concentrate on the third pillar mentioned above, the migration away from 80-bit security for public key cryptography. NSA restricts the use of public key cryptography in "Suite B" to elliptic curve cryptography (ECC). Compliance will be a challenge for applications that do not use ECC yet, because implementing ECC is non-trivial for a variety of reasons – because it involves mathematical structures that are much harder to understand and implement than those involved in RSA (cf. [39]), but also because various higher level system decisions need to be made, the consequences of which not even the *cognoscenti* seem to agree upon (cf. [25]). Finally, once one understands all mathematics and has decided which type of elliptic curves to use, it is still a challenge to figure out the best computational approach given one's platforms.

Where "Suite B" adopts 256-bit ECC as the lowest strength asymmetric system, NIST's "SP 800-57 (1)" allows 224-bit ECC and, more remarkably, continued use of RSA, but with 2048-bit moduli. Estimates vary as to what level of security is offered by 2048-bit RSA moduli. For the intended 112-bit security level, 2048-bit RSA may be regarded as satisfactory,

depending on one's view on the security provided by 1024-bit RSA (cf. below). The security level of any of the newly standardized systems is, however, not the subject of this paper – indeed, we will not even try to define what precisely is meant by 'security level' (but see the remark at the end of this paper). Instead, we address a short term but nevertheless important question that many practitioners will face the next few years, namely until when the current standards, 1024-bit RSA and 160-bit ECC, can responsibly be used: do businesses indeed have to adopt the new standards by the end of 2010, or can they allow for a smoother transition? For RSA this question, which includes the important practical concern by when digital certificates based on 1024-bit RSA should be phased out, is addressed in Section 2, for ECC in Section 3. For ECC we limit ourselves to curves over prime fields, since that is the focus of NSA's recommendations as well: throughout this paper, ECC refers to ECC over a prime field.

With a question of this sort we must say something about the adversarial model. Throughout this write-up we restrict our attention to work that has been or may be performed in the open research community. Such activities tend to make public potential weaknesses, which leads to undesirable liabilities for users of affected systems. Potential vulnerability to well-funded efforts that may take place behind closed doors, such as at government laboratories, is perceived in a different manner. As further argued below, the systems at hand are known to be already vulnerable to truly determined, large-scale attack efforts, but nevertheless their current usage is not generally considered to be "irresponsible." Therefore, when we say that a system can "responsibly be used", we mean that it must be considered unlikely that an academic, open community effort will be able to mount a successful attack. Obviously, this is by no means a formal definition and leaves many issues unaddressed (such as timing and funding level). Nevertheless, despite this vagueness, we strongly believe that a paper of this sort is useful for the cryptographic user community. We present the relevant current cryptanalytic state of the art and, combined with past developments, give our view on what can and what cannot be expected in the near future. More importantly, though, we present these data so that the reader can develop an understanding of the effort involved in breaking RSA or ECC for relevant parameter choices and draw his or her own conclusions.

For the purposes of the present write-up, we may assume that precautions are taken to avoid common pitfalls when implementing RSA or ECC such as, for RSA, surveyed in [6] and more recently pointed out in [23] if inadequate padding is used. Thus, we may concentrate on attacks that put a firm upper boundary on the security of any implementation of RSA or ECC: attacks that are commonly referred to as *brute-force attacks* that attempt to factor the public RSA modulus or to compute a discrete logarithm in an elliptic curve group.

2 The security of 1024-bit RSA

Since the introduction of the RSA cryptosystem, the effectiveness of brute-force attacks has improved considerably, with noticeable impact on the choice of RSA modulus sizes. The most effective method that has been published, the *Number Field Sieve* (NFS, cf. [27]), is already more than 20 years old. Its development, impact, and properties relevant for the assessment of the security of 1024-bit RSA moduli are reviewed in this section. Since 1989, essentially nothing has happened in integer factorization, except for a seemingly endless stream of relatively small improvements. All these tweaks, particularly when combined, influence the effectiveness

of NFS, but none of them represents a major new idea or breakthrough: factoring-wise, research is at a disconcerting or reassuring standstill. Notable exceptions are polynomial time factoring on a quantum computer [41], so far without practical implications, and polynomial time primality proving [1].

The original version of NFS, now referred to as the *Special Number Field Sieve* (SNFS), was expressly invented to fully factor the ninth Fermat number, $F_9 = 2^{2^9} + 1$, and succeeded doing so in 1990 (cf. [28]). SNFS crucially relies on the peculiar, "special" shape of the numbers it can handle, such as F_9, and in particular cannot be used to factor RSA moduli. It took a few more years before SNFS was generalized to the somewhat slower method now known as NFS that *can* handle RSA moduli. The first public announcement of the NFS-factorization of a 512-bit RSA modulus was in 1999 (cf. [9]).

The latest record of SNFS factorization was that of the "special" number $2^{1039} - 1$, found in 2007 (cf. [2]). An NFS-effort to factor a 768-bit RSA modulus is currently underway and is expected to finish in 2010. It may turn out to be a new NFS record that would beat the current factoring record of the 663-bit, 200-digit RSA modulus RSA-200 by more than 100 bits (cf. [3]). It is expected that the 768-bit factorization, when finished, will have required about 10 to 15 times the resources required for $2^{1039} - 1$. This estimate is based on current practical findings, not on theoretical estimates or guesswork: both are unreliable when switching between SNFS and NFS. Extrapolating 768-bit NFS to 1024-bit NFS (as required for a 1024-bit RSA modulus) can be done using the heuristic asymptotic runtime estimate for NFS. Doing so in the usual "$o(1)$-less" fashion (cf. [30] and [26]), one finds that 1024-bit RSA moduli are about 1200 times harder to crack than 768-bit ones, the first example of which is "about" to be completed. We round this down to one thousand based on the argument that there is a practical discontinuity in the NFS runtime for 'small' numbers that is not adequately reflected in the asymptotic runtime estimate when the $o(1)$ is omitted. Numbers of 768 bits are relatively close to the lower end of the range where number fields of extension degree 6 are optimal, whereas 1024-bit numbers are fairly close to the middle of that same range. As a consequence, the $o(1)$-less estimate for 768 bits is low compared to the actual runtime, where for 1024 bits they can be expected to be much closer. Thus, it may be expected that the 1200 is too pessmistic and that one thousand is closer to reality. Furthermore, combining the numbers, it is estimated that a 1024-bit RSA modulus requires resources that are at most 10 to 15 thousand times larger than for the already completed $2^{1039} - 1$ SNFS effort.

It may be argued that it is a bit of a stretch to extrapolate NFS factoring estimates from 768 to 1024 bits. But it is supported by a similarly calculated and relatively speaking even larger extrapolation from 512 to 768 bits: "$o(1)$-less" theoretical runtime extrapolation suggests that 768-bit RSA moduli are about 6 thousand times harder to crack than 512-bit ones, whereas empirical evidence based on best efforts using current software and hardware, suggests an average figure of 4 to 8 thousand.

Below, we estimate the resources currently required to tackle a 1024-bit RSA modulus, assuming rough familiarity with the major steps of an (S)NFS factorization. This is followed by a discussion of the security of 1024-bit RSA moduli in the near and not too distant future.

Polynomial selection. The result of this step (needed just for NFS, not for SNFS) greatly influences the time spent on the later steps, but the amount of computation spent here is

small compared to the others: so far about 3% to 5% of the overall computation. It is fully parallelizable. There is still ample room for improvement in current methods (cf. [8]).

Sieving. Sieving allows full and virtually communication-free parallelization and can be run at any number of locations on any number of independent processors. It is and has always been the step that requires the bulk of the computation. For $2^{1039}-1$ it required about two hundred years of computing on a single 3GHz Pentium D core, for a 768-bit RSA modulus we have established that ten times that effort suffices. For a 1024-bit RSA modulus the sieving time extrapolates to 2 to 4 million core years.

For a 768-bit RSA modulus 2 gigabytes of RAM per core works nicely, half of it still works too but is noticeably less efficient. For 1024 bits, 2 gigabytes of RAM per core would cut it too closely, but memories of 32 or 64 gigabytes per multi-core or multi-threaded processor (or even per mainboard) would work. These estimates take into account current processor and memory access speeds and are based on the fact that memory requirements grow roughly with the square root of the sieving time.

Filtering. Transforming the sieving data into a matrix is one of the least challenging steps of a factorization and does not need to be discussed here. For a 1024-bit RSA modulus it will require a few hundred terabytes of disk space, which is relatively easy to manage compared to the previous and next steps.

Matrix. The matrix requires time and memory growing as fast as it is for sieving. Although for all factorization attempts to date the matrix time has been modest compared to the sieving time, it has always been more cumbersome due to the distinctly smaller degree to which this step allows parallelization. The first time the step was parallelized over multiple locations was for the factorization of $2^{1039}-1$ reported in [2], namely on two sets of clusters, one in Japan and one in Switzerland. Overall it took about 35 core years on a 3GHz Pentium D. For a 1024-bit RSA modulus between half a million and a million core years should suffice. For each tightly coupled cluster participating in the matrix step a combined memory of 10 terabytes should be adequate.

Final square root. The runtime of this step has always been insignificant compared to all other steps. There is no reason why it would be different for a 1024-bit RSA modulus.

We stress that the figures above are based not just on extrapolation of the data from [2], but on analysis and extrapolation of data of an ongoing 768-bit RSA modulus factorization effort for which the sieving has been completed. These figures thus represent the most accurate estimates published so far for a 'realistic' 1024-bit RSA modulus factoring effort.

We consider what it means for the security of 1024-bit RSA – now, for the next five years, and for the next decade. At this point in time a brute-force attack against 1024-bit RSA would require about two years on a few million compute cores with many tens of gigabytes of memory per processor or mainboard. Though substantial, this is not an inconceivably large effort. We conclude that the computation is in principle already feasible: processors with enough memory for the sieving do exist, and there are clusters that have the combined memory required for the matrix (though most will not have enough nodes). The total budget required is large, but smaller than what was spent on other dedicated scientific endeavors.

Nevertheless, as an "open community" effort a 1024-bit RSA factorization is currently out of reach. For how long will this be the case? As usual when we try to predict the cryptanalytic future, we study past trends and use those as a basis for our predictions. Thus, as factoring breakthroughs have not occurred for several decades, and polynomial time factoring

on an actual quantum computer (cf. [41]) still seems to be several decades away, both these possibilities are discounted.

Next considering special purpose hardware, according to [40] sieving for a 1024-bit RSA modulus can be done in a year for about US $10,000,000, plus a one-time development cost of about US $20,000,000, and with a comparable time and cost for the matrix. Although the cost figures look reasonable, and are in the same ballpark as those cited in [21] and [22], the feasibility of the wafer scale integration required by [40] may be questioned (cf. [15] and the references therein). But, in our opinion skepticism met by these designs is beside the point. The cost figures should not be interpreted as upper bounds, i.e., "Be careful, 1024-bit RSA can be broken in two years for about twenty million bucks (assuming free development)," but as lower bounds, i.e., "No reason to worry too much: even under very favorable assumptions, factoring a 1024-bit RSA modulus still requires enormous resources." The estimates should thus not be read as threatening but as confidence-inspiring since the required resources are out of reach of open community efforts as considered here.

The above points aside, even if one manages to build special purpose hardware, by that time commodity hardware already may have caught up, speed-wise or cost-wise – at least that has consistently been the case in the past. Peter Hofstee (cf. [18]) predicts that this will not change until at least 2030, at which point special purpose hardware may be the only way to stay on Moore's curve. That is in a future that is too distant for our purposes. According to others, however, this shift to special purpose hardware may never materialize (cf. [20]). Furthermore, as brought to our attention by Paolo Ienne, figures as presented in [21] and [22] suggest that hardware development and manufacturing costs as often found in cryptanalytic contexts tend to be on the optimistic (i.e., low) side.

Turning back to reality, the following events are relevant for our question when a 1024-bit RSA modulus can be expected to fall:

1988: First factorization of a hard 100-digit integer (cf. [29]).
1999: First factorization of a 512-bit RSA modulus (cf. [9]).
2010: Expected first factorization of a 768-bit RSA modulus.
20??: Expected first factorization of a 1024-bit RSA modulus.

Based on NFS runtime comparisons the three consecutive "difficulty gaps" between the above four factorizations are factors of about 2,000, 6,000, and 1,000. The last two numbers were already given above and the 6,000 was shown to be a close match to empirical evidence. It should be noted that the 100-digit factorization was not obtained using NFS but using its asymptotically slower predecessor 'Quadratic Sieve'. Thus, in reality, a larger hurdle than just a factor 2,000 was overcome in the period 1988-1999, mostly due to the invention and development of NFS starting in 1989. In any case, the first three entries above suggest that a thousand-fold increase in factoring difficulty can traditionally be overcome in a decade. This reasoning would naively lead to an expected first factorization by the open community of a 1024-bit RSA modulus approximately by the year 2020. But is it reasonable to expect that the developments that took place between 1988 and now will continue in the decades to come?

To explain the factor of more than one thousand per decade – as opposed to just observing it twice in a row in the past – a factor of one hundred every ten years can fairly consistently be attributed to Moore's law. The other factor is due to lots of gradual improvements in

the polynomial selection step and the sieving and matrix software, probably combined with greater persistence and patience among the researchers involved. The type of compute power required for large scale NFS efforts will keep following Moore's law at least for the next two or three decades (cf. [18], [20], [21]), and realistic progress on algorithmic fronts has been announced as well (cf. [8]). Thus, there is enough reason to expect that over the next decade we will be able to overcome yet another factor of one thousand in factoring difficulty.

This explanation of the observed progress bolsters the confidence in the above naive prediction that in about 10 years the first open community 1024-bit RSA modulus factorization can be expected – assuming by then volunteers can be found to take up the considerable challenge. In the next paragraph, and to conclude this section, we consider the question whether we can expect a 1024-bit RSA modulus factorization much sooner.

Assume that the 768-bit RSA modulus factoring attempt will be completed in the year 2010. The factor of one hundred provided every decade by Moore's law implies a factor of ten every five years. Thus, to be able to factor a 1024-bit RSA modulus in 2015 (= 2010 + 5) the combined algorithmic improvements would have to accumulate to a factor of about one hundred well within the next five years. Such progress would be unprecedented unless a major breakthrough occurs. Given the unlikelihood of the latter, an open community effort that would factor a 1024-bit RSA modulus cannot be expected by the year 2015. This conclusion is compounded by the observation that the current 768-bit effort already stretches the resources of open community factoring enthusiasts: it is virtually inconceivable that a thousand-fold larger effort would be feasible in just five years. Obviously not a hard argument, but a more intuitive one – steeped in decades of leading edge integer factorization experience.

3 The security of 160-bit ECC

The most effective brute-force method to compute a discrete logarithm in a prime order elliptic curve group is Pollard's rho method (cf. [38]). We assume that this group order and the order of the underlying prime field have the same bitlength (as is for instance the case for the curves in [33, Section D.1.2]). Pollard rho comes with a whole lot of embellishments, some with major, others with relatively minor impact. Most importantly, the method allows full parallelization on any number of very loosely coupled processing elements that require only occasional communication with a central server (cf. [44]). Denoting the group order by q, each Pollard rho process needs memory for a small constant number of $(\log_2 q)$-bit numbers. Thus – and this in huge contrast to NFS-based brute force attacks against RSA – virtually any processing element can, in one way or another, contribute to an ECC attack. Furthermore, if a processing element allows multi- or hyperthreading or any other fancy type of parallelization or pipelining, this can in principle be taken advantage of by running as many Pollard rho processes in parallel as required to keep the device fully occupied.

This very 'lightweight' parallelization changes the applicable threat model compared to our estimates of the resources required to attack 1024-bit RSA. In particular we need to examine whether the most recently published result on ECC brute-force attacks still adequately represents current possibilities: in 2002 it took "10^4 computers (mostly PCs) running 24 hours a day for 549 days" to solve an elliptic curve discrete logarithm problem over a 109-bit prime field (cf. [11]). The large team performing this calculation used Chris Monico's Pollard rho implementation, with highy optimized assembly code tuned to a wide variety of PC platforms.

Naively accommodating for 5 'Moore doublings' over a period of 5 times 18 months, but increasing the difficulty by 100M (which approximates $\left(\frac{160}{109}\right)^2 \cdot 2^{(160-109)/2}$, taking into account that the number of group operations grows with the square root of the group size, combined with the quadratically growing group operation cost itself) Monico's result extrapolates to about 50 billion years to attack 160-bit ECC on a current core. Obviously, this would be a daunting task. Our question is if this estimate is valid given current multi-core processors, and how the required computational effort can be realized in the most economical fashion. For this purpose we consider, in the remainder of this section, a number of different platforms that may be used to attack 160-bit ECC using Pollard's rho method.

Pollard's rho method was designed to calculate discrete logarithms in multiplicative groups of prime fields. We already mentioned that it allows ideal parallelization. This is based on the use of *distinguished points* as described in [44]. Furthermore, the *negation map* (cf. [14], [45]), *adding walks* (cf. [43]) and *tag-tracing* (cf. [13]) make the calculation somewhat faster. As described in [13] tag-tracing does not immediately apply to elliptic curve groups, but has been shown in [7] to give a moderate speed-up there as well. According to [7], tag-tracing does not seem to be compatible with usage of the negation map or, more in general, the use of equivalence classes (applicable mostly to curves over small characteristic extension fields, and thus of no concern here). For elliptic curves over prime fields each of these improvements does not affect the overall expected runtime by more than 40%, and is therefore hardly of concern for our primary purpose to get an order of magnitude estimate.

With elliptic curves there is the issue how to represent group elements: affinely requiring only a few multiplications plus an inversion (all in the prime field) per group operation, or projectively to avoid the inversion at the cost of more multiplications. Recognition of distinguished points, however, enforces a unique representation and thereby an inversion. In circumstances such as the present one where many processes can be run simultaneously, their inverses can be shared at the cost of, roughly, 3 additional multiplications per curve, using the simultaneous inversion from [31]. As a result, when processing N processes in parallel and using affine Weierstrass representation, point addition takes four multiplications, one squaring and $\frac{1}{N}^{\text{th}}$ inversion (all in the prime field) for the x-coordinate, plus one multiplication for the y-coordinate.

For a variety of platforms, we consider best-effort implementations, either from the literature or derived by ourselves, of Pollard's rho method in elliptic curve groups, while making use of full parallelization, adding walks, affine point representation with simultaneous inversion, and possibly other improvements if compatible with the platform and other implementation aspects. The platforms in question, in rough order of user-friendliness, are regular desktops or servers, Sony PlayStation 3 game consoles (PS3), graphics cards, FPGAs, and ASICs.

Desktop. The fastest actually working desktop implementation of Pollard rho on elliptic curves that we are aware of is reported in [7]. It focuses on the 131-bit prime field ECC challenge from [10], implemented as indicated above and processing 200 walks per core. For generic 131-bit primes, an expected overall runtime of 10^5 years is reported on an AMD Phenom 9500 quad-core 64-bit processor with a clock frequency of 2.2GHz, i.e., about $4 \cdot 10^5$ core years. A 25% speed-up can be obtained due to the particular size of the 131-bit prime used. Both the regular and the speeded-up implementation use assembly code. It should be noted that [7] uses tag-tracing as opposed to the negation map. Swapping these may lead to an overall speedup of about 20%.

Extrapolation of the generic runtime to 160-bit primes, in the same manner as the runtime reported by Chris Monico was extrapolated, yields $\left(\frac{160}{131}\right)^2 \cdot 2^{(160-131)/2} \cdot 4 \cdot 10^5 \approx 10^{10}$ core years. We stress that this runtime is realistic and that 2.5 billion current workstations should be able to crack 160-bit ECC within a time-frame of 6 to 24 months (i.e., give or take a factor of 2, depending on which embellishments are used or which unexpected snags will occur). This is about 5 times faster than what naive extrapolation of Monico's result made us believe, but still not a task that is feasible anytime soon for an open community effort – indeed, even for a well sponsored agency it would be a formidable task. Below we consider whether it can be done 'cheaper.' The factor 5 may be attributed to the fact that our type of application heavily relies on integer multiplication which is, relatively speaking, much faster on the current 64-bit processors than on the 32-bit ones from back in 2002.

PS3. The Sony PlayStation 3 game console is powered by a Cell processor, a powerful unit that is described in more detail below. The Cell's processing power can be unlocked for general applications, i.e., not just games, using Sony's hypervisor, thereby making the PS3 a relative inexpensive source of processing power. Actually using this processing power, however, in particular for the type of application that we are interested in here, is not straightforward: it is hard to optimize multi-precision integer arithmetic on regular processors, it is not easier on the peculiar Cell architecture. A first approach is to use IBM's off-the-shelf MPM library for long integer arithmetic (cf. [19]). Unfortunately the performance did not live up to what we had expected given the Cell's specifications, lagging behind by about an order of magnitude.

Thus, we set out to design our own implementation, geared toward optimal performance for our application to Pollard rho for elliptic curve groups over relatively small prime fields. We give a rough outline of the design of our arithmetic functions. For further details we refer the interested reader to the Appendix.

Omitting many details, the Cell has a main processing unit (the PPE, a dual-threaded 64-bit Power Processing Element) which can offload work to eight Synergistic Processing Elements (SPEs). The for us relevant part of the SPE is the Synergistic Processing Unit (SPU). Each SPU runs independently of the others at 3.2 GHz and works on its own 128 registers of 128 bits each and has its own 256 kilobyte Local Store for its instructions and data. The 128-bit registers allow SIMD operation on sixteen 8-bit, eight 16-bit, or four 32-bit integers. Finally, the SPU has two pipelines, an odd and an even one, which means that two instructions (one odd, one even) can be dispatched per clock cycle per SPU. Although it is rich in boolean operations, its integer multiplication facilities are somewhat limited, cf. Appendix. There is no smart branch prediction on the SPU. It is therefore advisable, as customary in SIMD anyhow, to avoid branching as much as possible.

Of the 8 SPEs one is disabled and one is reserved by Sony's hypervisor. Thus, when running Linux (under the hypervisor), 6 of the 8 SPEs are accessible. Because each SPU can be regarded as a four-way SIMD 32-bit processor (with limited multiplication possibilities) and runs at 3.2 GHz, it may under favorable circumstances be possible to squeeze performance equivalent to $4 \times 6 = 24$ 'regular' (but 32-bit) cores out of a single PS3. As reported in [42], sometimes even more is possible. We did not manage to achieve this for our elliptic curve Pollard rho application, but we got quite close, as implied by the numbers below.

More precisely, we obtained the following results. We focussed our attention on *curve number 6* from [12], which is a curve over a 112-bit prime field and which is, as of 2009, the 'smallest'

elliptic curve standardized for cryptographic purposes. To minimize the inversion overhead, we processed as many walks per SPU as permitted by the size of the Local Store. This turned out to be 400 for the 112-bit prime field, thus we could use $N = 400$ for the simultaneous inversion. The 400 walks are organized in 100 sequential (but synchronized at inversion time) batches of 4 walks that are processed simultaneously by performing the underlying field arithmetic in a 4-way SIMD fashion. It turned out that for the specific 112-bit prime certain computational advantages could be obtained (cf. Appendix) that do not translate to generic primes of the same size. For generic primes the arithmetic is at most 20% slower.

For the special 112-bit curve we found that the overall calculation can be expected to take approximately 60 PS3 years[4] . Since we use just 6 SPUs, better performance can be obtained if we would use the PPU too, or if we would be able to access the 7^{th} SPU. Extrapolating while including a 20% generic slowdown (and also accounting for a lower value of N for the larger prime), we find $\left(\frac{160}{112}\right)^2 \cdot 2^{(160-112)/2} \cdot 1.2 \cdot 60 \approx 2.5 \cdot 10^9$ PS3 years to crack 160-bit ECC. Comparing to the number of desktops calculated earlier, we again find that a year on 2.5 billion devices suffices. For the current application 6 SPUs are computationally apparently more or less equivalent to a quad-core 64-bit processor, i.e., a single SPU is equivalent to 66% of a single 64-bit core. The latter is better at integer multiplication, which outweighs, for this application, the advantage of the 4-way 32-bit SIMD SPU architecture. As mentioned above, for other types of applications the advantage may be on the other side.

It is debatable which of the two options – desktops or PS3s – is more economical. The relevant parts for a desktop can be purchased for about the same price as a PS3, but the latter is ready to go and the former needs to be assembled, billions of times if one gets serious about an ECC attack. The PS3 comes with a Blu-ray player that is not needed, but is attractively priced just because of it, so getting the non-assembled relevant parts of a PS3 may actually turn out to be more expensive (cf. relatively high cost of Cell-blades). We leave quantum-discounts for an order of several billion devices of either type to the reader's negotiation skills. Note that the PS3 also comes with a rather powerful graphics card which is unfortunately, with the present version of the hypervisor, not accessible to applications other than gaming. If it were accessible, it could double the effectiveness of a PS3 (cf. below). As is, the balance may already tip in favor of the PS3, compared to a desktop, if we would also involve the PS3's PPE, and in particular the PPE's AltiVec SIMD instruction set, in the calculation. This is under investigation.

We refer to the Appendix for some of the details of our implementation and in particular for an interesting 4-way SIMD modular inversion method that is ideally suited to the SPU. An interesting aspect of our implementation is that we allow occasional errors by omitting the tests and branching that would have caught and fixed them. This implies that of the many parallel streams that we are processing, occasionally one may produce an erroneous result. Obviously, this would be recognized and the result thrown out – except that despite months of computations on many machines it has not occurred yet. Overall, we gain a very considerable speed-up using this approach.

Graphics card. As far as we can tell at this point, graphics cards do not dramatically change the above picture yet. In [4] it is reported that a GTX 295, containing two graphics processing units, can perform about 42 million modular multiplications per second for 280-bit moduli.

[4] As reported on `http://lacal.epfl.ch/page81774.html` this calculcation was completed on July 8, 2009, taking the predicted amount of time. A description of the full details of the computation is in preparation.

Extrapolating the generic SPU modular multiplication times from the Appendix to this same modulus size (cf. last paragraph of Section A.1), we find that a single SPU can do almost 8 million modular multiplications per second for 280-bit moduli, for a total of 47 million per second per PS3. These figures are comparable, and we are not aware of a reason why that would not be the case for 160-bit moduli. The price and power consumption of graphic cards, however, both look less attractive than those of a PS3.

FPGA. For FPGAs we consult [16, Table V]: using a single XC3S1000 chip it would take about $3.1 \cdot 10^{11}$ years to crack 160-bit ECC. Furthermore, according to [16, Section 6.1], this time can be reduced by a factor 120 to $2.6 \cdot 10^9$ years when 120 of such chips are purchased for US$10,000. Note that $2.6 \cdot 10^9$ is close to the number of PS3 years required, as analysed above. Also note that about 25 PS3s can be purchased for US$10,000. It follows that FPGAs are not competitive. This hardly changes if we allow for a 'Moore factor' of about 2 in favor of FPGAs (since [16] was published in 2007).

ASIC. All performance figures presented so far are based on actual implementations or realistic extrapolations thereof. For ASICs we have not been able to find performance figures that are backed up by actual implementations. Nevertheless, from [16, Table VII] we conclude that ASIC performance is estimated to be about 200 times more economical than FPGAs for 160-bit ECC. Combined with the above factor 25 in favor of PS3s over FPGAs, we find that price-wise ASICs should outperform PS3s by about a factor 8. Thus, for about an eighth of the cost of 2.5 billion PS3s, one should be able to crack 160-bit ECC in about a year. We argue, as in our discussion of 1024-bit RSA backed up by [18], [20], [21], and [22], that for the type of efforts considered here the time is not yet ripe for special purpose hardware, and do not consider ASICs in our analysis below. It should be noted, though, that large scale special purpose hardware attacks against ECC do not require wafer scale integration and thus look more feasible than the designs proposed in [40]: in principle the calculation is certainly doable.

From the above we conclude that, computationally speaking, cracking 160-bit ECC is at least three orders of magnitude harder than cracking 1024-bit RSA. For 1024-bit RSA we argued that the open community may have a chance by the year 2020, but that before 2015 nothing can be expected. The RSA-estimates take into account continuation of the usual steady stream of algorithmic improvements, contributing a factor 10 over a decade. For ECC the contribution of algorithmic improvements has been smaller. Even if they occur at the same rate as for RSA, we are still looking at the same difficulty gap of a factor one thousand by the year 2020, i.e., the year that cracking 1024-bit RSA may be feasible for the open community.

On the other hand, running a whole lot of Pollard rho processes is quite a bit less cumbersome than running the NFS sieving and matrix steps. Indeed, Chris Monico for his 109-bit ECC effort back in 2002 managed to attract a number of contributing computers that is about an order of magnitude larger than the current large scale factoring effort for a 768-bit RSA modulus. Given the large number of PS3s 'out there,' can the required number of compute cycles not conveniently be harvested using some type of PS3-BOINC project (cf. [5])? Right now most certainly not: 2.5 billion PS3s or equivalent devices (such as desktops) for a year is way out of reach. In a decade, very optimistically incorporating 10-fold cryptanalytic advances, still millions of devices would be required, and a successful open community attack on 160-bit ECC even by the year 2020 must be considered very unlikely.

4 Conclusion

Moving from 80-bit security to one of the proposed higher security levels is a good idea. There is, however, no reason to be concerned that usage of 1024-bit RSA until the year 2014 carries a major risk. For instance, there is no indication that certificates relying on 1024-bit RSA will have to be revoked so long as they naturally expire by 2014. Implications for other use cases of 1024-bit RSA are easy to infer. Similarly, there does not seem to be any reason to be concerned about continued usage of 160-bit prime field ECC during the next decade.

Recommendations of this sort can be made by anyone, in particular if they are, as the one concerning 1024-bit RSA, not particularly surprising. The value of our work is, however, that we are deeply involved in all the new theoretical and practical algorithmic research and grunt work required to be confident to make them. Although we are simply driven by algorithmic curiosity and interest in the underlying mathematical and computational problems, we are also willing to stick out our necks to interpret and publish the likely practical implications of our activities, not in our own immediate interest but of potential use to the RSA and ECC user communities. We remark that our 1024-bit RSA estimate supports the recommendation proposed in [35]. Our estimates may turn out to be controversial – and may trigger out of the ordinary efforts to prove them wrong.

Remark on security levels. Defining "56-bit security" as the level of protection offered by single DES (cf. [30] and [26]), we conclude from [37] that 56-bit security can on average be broken on a single PS3 (using 6 SPUs) in about 1.75 year. Combining that figure with the above PS3 effort to crack 160-bit ECC and observing that $2^{30} < 2.5 \cdot 10^9/1.75 < 2^{31}$, we find that 160-bit ECC provides between 86 and 87 bits of security. It would be reasonable to conclude that 1024-bit RSA provides about 76-bit security. The figures for ECC in [30, Table 1] are similar, but 1024-bit RSA looks a bit more secure than suggested there: apparently improvements in NFS have not lived up to the expectations from [30]. Note that this 10-bit gap allows one to use 160-bit ECC with a 10-bit cofactor (as in [33, Table 1] but as explicitly excluded here) without getting security less than 1024-bit RSA.

Acknowledgements

We gratefully acknowledge discussions with Paolo Ienne, comments by Chris Babel and his team and by Paul Hoffman, and a number of anonymous comments. This work was supported by the Swiss National Science Foundation under grant numbers 200021-119776 and 206021-117409 and by EPFL DIT. Travel related to the activities described in Section 2 was supported by the European Commission through the EU ICT program ECRYPT II.

References

1. M. Agrawal, N. Kayal, N. Saxena, Primes is in P, Annals of Mathematics 160, pp. 781–793 (2004).
2. K.Aoki, J. Franke, T. Kleinjung, A.K. Lenstra, D.A. Osvik, A kilobit special number field sieve factorization, Asiacrypt 2007, LNCS 4833, pp. 1–12, 2007.
3. F. Bahr, M. Boehm, J. Franke, T. Kleinjung, Factorization of RSA-200, May 2005, `http://www.loria.fr/~zimmerma/records/rsa200`.
4. D.J. Bernstein, T.-R. Chen, C.-M. Cheng, T. Lange, B.-Y. Yang, ECM on graphics cards, Eurocrypt 2009, LNCS 5479, pp. 483–501, 2009.

5. BOINC, Open-source software for volunteer computing and grid computing. http://boinc.berkeley.edu/.
6. D. Boneh, Twenty years of attacks on the RSA cryptosystem, Notices of the American Mathematical Society, 46: 203–213, 1999. http://crypto.stanford.edu/~dabo/papers/RSA-survey.pdf.
7. J.W. Bos, M.E. Kaihara, T. Kleinjung, Pollard rho on elliptic curves, manuscript, submitted for publication, May 2009.
8. CADO workshop on integer factorization, presentations by T. Kleinjung, P.L. Montgomery, J. Papadopoulos, P. Stach, and others, Nancy, France, October 7–9, 2008: http://www.sigsam.org/bulletin/issues/issue167.html.
9. S. Cavallar, B. Dodson, A.K. Lenstra, P. Leyland, P.L. Montgomery, B. Murphy, H. te Riele, P. Zimmermann, et al., Factoring a 512-bit RSA modulus, Proceedings Eurocrypt 2000, Springer-Verlag LNCS 1807, 1–18, 2000.
10. Certicom. Certicom ecc challenge. http://www.certicom.com/images/pdfs/cert_ecc_challenge.pdf, 1997.
11. Certicom. Press release: Certicom announces elliptic curve cryptosystem (ECC) challenge winner. See http://www.certicom.com/index.php/2002-press-releases/38-2002-press-releases/340-notre-dame-mathematician-solves-eccp-109-encryption-key-problem-issued-in-1997, 2002.
12. Certicom Research. Standards for efficient cryptography 2: recommended elliptic curve domain parameters. Standard SEC2, Certicom, 2000.
13. J.H. Cheon, J. Hong, M. Kim, Speeding up the Pollard rho method on prime fields, Asiacrypt 2008, LNCS 5350, pp. 471–488 (2008).
14. R.P. Gallant, R.J. Lambert, and S.A. Vanstone, Improving the parallelized pollard lambda search on anomalous binary curves, Math. Comp. 69, pp. 1699–1705 (2000).
15. W. Geiselmann, R. Steinwandt, Non-Wafer-Scale Sieving Hardware for the NFS: Another Attempt to Cope with 1024-bit, Eurocrypt 2007, LNCS 4515, pp. 466–481 (2007).
16. T. Güneysu, C. Paar, J. Pelzl, Special-Purpose Hardware for Solving the Elliptic Curve Discrete Logarithm Problem, ACM Trans. Reconfigurable Technol. Syst., pp. 1–21 (2008).
17. K. Hickey, The cutting edge of defense IT, Encrypting the future, Government Computer News, August 2007: http://www.gcn.com/print/26_20/44796-5.html.
18. P. Hofstee, Heterogeneous Multi-core Processors, public lecture at EPFL, September 19, 2008.
19. IBM, Example Library API Reference, Version 3.0, http://www.ibm.com/developerworks/power/cell/documents.html, 2007.
20. P. Ienne, personal communication, August 2009.
21. International technology roadmap for semiconductors, 2007 edition, design, http://www.itrs.net/Links/2007ITRS/2007_Chapters/2007_Design.pdf.
22. International technology roadmap for semiconductors, 2008 update, overview, http://www.itrs.net/Links/2008ITRS/Update/2008_Update.pdf.
23. A. Joux, D. Naccache, E. Thomé, When e-th roots become easier than factoring, Asiacrypt 2007, LNCS 4833, pp. 13–28 (2007).
24. B.S. Kaliski, Jr., The Montgomery inverse and its applications, IEEE Trans. Computers, 44, pp. 1064–1065, 1995.
25. A.H. Koblitz, N. Koblitz, A. Menezes, Elliptic curve cryptography, the serpentine course of a paradigm shift, September 2008, eprint.iacr.org/2008/390.
26. A.K. Lenstra, Unbelievable security, Proceedings Asiacrypt 2001, Springer-Verlag LNCS 2248, 67–86.
27. A.K. Lenstra, H.W. Lenstra, Jr. (editors), The development of the number field sieve, Springer-Verlag LNM 1554, August 1993.
28. A.K. Lenstra, H.W. Lenstra, Jr., M.S. Manasse, J. Pollard, The factorization of the ninth Fermat number, Math. Comp. 61: 319–349, 1994.
29. A.K. Lenstra, M.S. Manasse, Factoring by electronic mail, Proceedings Eurocrypt 1989, Springer-Verlag LNCS 434, 355–371, 1989.
30. A.K. Lenstra, E.R. Verheul, Selecting cryptographic key sizes, J. of Cryptology 14: 255–293, 2001.
31. P.L. Montgomery, Speeding the Pollard and elliptic curve methods of factorization, Math. Comp. (1987), pp. 519–521.
32. National Institute of Standards and Technology, Suite B Cryptography: http://csrc.nist.gov/groups/SMA/ispab/documents/minutes/2006-03/E_Barker-March2006-ISPAB.pdf.
33. National Institute of Standards and Technology, FIPS Pub 186-3, Digital Signature Standard (DSS), http://csrc.nist.gov/publications/fips/fips186-3/fips_186_3.pdf.

34. National Institute of Standards and Technology, Special Publication 800-57: Recommendation for Key Management Part 1: General (Revised), http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf.
35. National Institute of Standards and Technology, Discussion paper: the transitioning of cryptographic algorithms and key sizes, http://csrc.nist.gov/groups/ST/key_mgmt/documents/Transitioning_CryptoAlgos_070209.pdf.
36. National Security Agency, Fact sheet NSA Suite B Cryptography: http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml.
37. D.A. Osvik, E. Tromer, Cryptologic applications of the PlayStation 3: Cell SPEED: http://www.hyperelliptic.org/SPEED/slides/Osvik_cell-speed.pdf.
38. J.M. Pollard, Monte Carlo methods for index computation (mod p), Math. Comp (1978), pp. 918–924.
39. R. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public key cryptosystems, Commun. of the ACM, 21: 120–126, 1978.
40. A. Shamir, E. Tromer, Factoring large numbers with the TWIRL device, Proceedings Crypto 2003, Springer-Verlag LNCS 2729, 1–26, 2003.
41. P.W. Shor, Algorithms for quantum computing: discrete logarithms and factoring, Proceedings of the IEEE 35th ann. symp. on foundations of computer science, 124–134, 1994.
42. M. Stevens, A. Sotirov, J. Appelbaum, A.K. Lenstra, D. Molnar, D.A. Osvik, B. de Weger, Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate, http://eprint.iacr.org/2009/111, Crypto 2009, LNCS 5677, pp. 55–69 (2009).
43. E. Teske, On random walks for Pollard's rho method, Math. Comp. (2001), pp. 809–825.
44. P.C. van Oorschot and M.J. Wiener, Parallel collision search with cryptanalytic applications, Journal of Cryptology 12 (1999), pp. 1–28.
45. M.J. Wiener and R.J. Zuccherato, Faster attacks on elliptic curve cryptosystems, SAC 1998, LNCS 1556, pp. 190–200 (1998).

A Multi-precision integer arithmetic on the SPU

We present some of the details of our SPU multi-precision integer arithmetic implementation on the PS3 SPU. This implementation is designed for the relatively short integers of at most a few hundred bits as used for ECC over prime fields. For these sizes we can, given the SPU architecture and because our Pollard rho application allows any level of parallelism, profit from 4-way SIMD parallelism by processing four identical or similar long integer operations simultaneously. For applications requiring larger integers (such as RSA) or where parallelism cannot be used, entirely different implementations are required. Those have been developed as well, but will not be described here.

A.1 Representation of relatively short long integers

Each SPU has 128 registers of 128 bits each. Each 128-bit SPU register v can be interpreted as a vector of eight 16-bit values $(v_{0,0}, v_{0,1}, v_{1,0}, \ldots, v_{3,1})$ or as a vector of four 32-bit values (v_0, v_1, v_2, v_3). When doing modular multiplication, n registers $X[0], X[1], \ldots, X[n-1]$ are interpreted as four $16n$-bit integers $X_{0,h}, X_{1,h}, X_{2,h}, X_{3,h}$ for $h \in \{0, 1\}$ with $X_{j,h} = \sum_{i=0}^{n-1} X[i]_{j,h} 2^{16i}$ with the understanding that if $h = 0$ then the $X[i]_{j,1}$ are zero and if $h = 1$ then the $X[i]_{j,0}$ are zero. While doing modular subtraction or inversion, m registers $X[0], X[1], \ldots, X[m-1]$ are interpreted as four $32m$-bit integers X_0, X_1, X_2, X_3 with $X_j = \sum_{i=0}^{m-1} X[i]_j 2^{32i}$. It is understood that any four-tuple $[A, B, C, D]$ of $16n$-bit or $32n$-bit integers can be represented in this way using n 128-bit registers consisting of four *short* integers (with $h = 0$ or $h = 1$) or m 128-bit registers consisting of four *long* integers, respectively.

SIMD instructions are then applied to the four-tuple $[A, B, C, D]$ by applying the relevant SIMD instructions to the corresponding n or m registers.

The short representation is necessitated by the fact that the SPU has just a $16 \times 16 \rightarrow 32$-bit 4-way SIMD multiplier, the long one is to take advantage of fast 32-bit 4-way SIMD operations (such as addition and subtraction). Because switching back and forth between the two representations is relatively easy using shuffle operations (which rearrange bytes), and because furthermore the shuffle operations can be dispatched almost for free since they are odd pipeline instructions as opposed to the even pipeline arithmetic ones, this turned out to be the most efficient approach.

Thus, the 128-bit register width implies that four long integers can be processed simultaneously in SIMD mode, but this fixed width does not impact the efficiency when dealing, for different applications, with equal length moduli of another size: all one needs to do is use a different value for n: for instance, $n = 4$ for a 128-bit modulus, and $n = 5$ for a 160-bit one. When moving to larger moduli, various overheads will have a relatively smaller effect on the cycle count, so that the usual quadratic cost extrapolation for schoolbook multiplication and such will be on the pessimistic side.

A.2 Modular arithmetic

Let $R = 2^{128}$ and $\tilde{p} = R - 3$. Curve number 6 from [12] is defined over the prime field of characteristic p, where p is the 112-bit prime $\tilde{p}/(11 \cdot 6949)$. We show that it is computationally advantageous to use a redundant representation modulo $\tilde{p}$ as opposed to p, while only reducing results to a unique representation in $\{0, 1, \ldots, p-1\}$ when so required (cf. Section A.4).

Let $\mathfrak{R}(x) = x \bmod R + 3\left\lfloor \frac{x}{R} \right\rfloor$ for non-negative x, then $x \equiv \mathfrak{R}(x) \bmod \tilde{p}$. Furthermore, with high likelihood $\mathfrak{R}$ can be used to quickly reduce values modulo $\tilde{p}$. Because $0 \leq \mathfrak{R}(x) < 4R$ for any x with $0 \leq x < R^2$, it follows that $0 \leq \mathfrak{R}(\mathfrak{R}(x)) < R + 9$. It is easily seen that $R + 9$ can be replaced by $R + 6$. Assuming that all values have more or less the same probability to occur, the result will actually most likely be $< \tilde{p}$. Although counterexamples are simple to construct and we have no formal proof, we can confidently state the following.

Proposition 1. *For independent random 128-bit non-negative integers x and y there is overwhelming probability that $0 \leq \mathfrak{R}(\mathfrak{R}(x \cdot y)) < \tilde{p}$.*

Thus, our modular multiplication works as follows. We have two four-tuples of 128-bit integers $[X_0, X_1, X_2, X_3]$ and $[Y_0, Y_1, Y_2, Y_3]$, where each tuple is represented using four 128-bit registers of 4 long (i.e., 32-bit) integers per register. Using shuffle operations the corresponding short (i.e., 16-bit) representations are extracted, per tuple requiring eight 128-bit registers (of 4 short integers per register). Next, the 4-way SIMD multiply-and-add instructions are used along with ordinary schoolbook multiplication to compute a four-tuple of 256-bit integers $[Z_0, Z_1, Z_2, Z_3]$, represented using eight 128-bit registers of 4 long integers per register, such that $Z_i = X_i \cdot Y_i$ for $i = 0, 1, 2, 3$. Multiply-and-add works nicely, because addition of a short value d to the product of shorts a and b while including a short carry c does not cause overflow: if $0 \leq a, b, c, d < 2^{16}$, then $d + (a \cdot b) + c < 2^{32}$. The product four-tuple $[Z_0, Z_1, Z_2, Z_3]$ is then reduced modulo $\tilde{p}$ by twice applying $\mathfrak{R}$ in 4-way SIMD fashion (cf. Prop. 1), where in most cases the second application of $\mathfrak{R}$ requires only a single multiply-and-add.

Algorithm 1 4-SIMD Extended Binary GCD

Input: $r = 2^{32}$, $p : r^{n-1} < p < r^n$ and $\gcd(p, 2) = 1$
$x : 0 < x < r^n$ and $\gcd(x, p) = 1$
Output: $z \equiv \frac{1}{x} \bmod p$
Algorithm:

```
[A1, B1, A2, B2] := [p, 0, x, 1] and [k1, k2] := [0, 0]
while (true)
    /* Start of shift reduction. */
    Find t1 such that 2^t1 | A1
    Find t2 such that 2^t2 | A2
    [k1, k2] := [k1 + t1, k2 + t2]
    [A1, B1, A2, B2] := [A1 >> t1, B1 << t2, A2 >> t2, B2 << t1]

    /* Start of subtraction reduction. */
    if (A1 > A2) then
        [A1, B1, A2, B2] := [A1 − A2, B1 − B2, A2, B2]
    elseif (A2 > A1) then
        [A1, B1, A2, B2] := [A1, B1, A2 − A1, B2 − B1]
    else
        Return z := B2 · (2^−(k1+k2)) mod p
    endif
endwhile
```

Our implementation takes care to fill both the even and the odd pipelines, thereby considerably reducing the overall latency. The average number of clock cycles required per 4-way SIMD multiplication modulo $\tilde{p}$ is about 215, i.e., 54 clock cycles per Pollard rho process.

Modular subtraction also uses the 4-way SIMD approach and can easily be made to work on the long representations, i.e, just four 128-bit registers to represent a four-tuple of 128-bit integers. Expensive branching is avoided by appropriate use of mask instructions. Overall, a 4-way SIMD subtraction modulo $\tilde{p}$ takes 16 instructions in the odd and 20 in the even pipeline, for a total of 20 clock cycles. This becomes 5 clock cycles per Pollard rho process.

A.3 Modular inversion

The calculation of the modular inverse of a positive integer x in the residue class of the odd modulus p is done by the algorithm depicted in Alg. 1. The method used is the one as described in [24], as it is very suitable for implementation on the 4-way SIMD architecture of the SPU. It proceeds in two phases:
1) The computation of the almost Montgomery inverse $x^{-1} \cdot 2^k \bmod p$ for some k.
2) A normalization phase where the factor $2^k \bmod p$ is removed.
Because 4 variables are employed on which operations can be carried out in SIMD-mode, they are grouped together as $[A_1, B_1, A_2, B_2]$, as set forth in Section A.1. In the algorithm, $X >> t$ ($X << t$) means that variable X is shifted by t bits towards the least (most) significant bit position (the SPU lets each of the elements of a four-tuple have its own shift amount). Note that, in the algorithm, operations $A_1 >> t_1$ and $A_2 >> t_2$ shift out only zero bits. The arithmetic is unsigned.

Let $g = \gcd(x, p)$. Let y be a solution of $xy \equiv g \pmod{p}$. The algorithm has invariants

$$\begin{aligned} k_j \geq 0, \quad A_j > 0, \\ A_j(2^{k_1+k_2}y) \equiv B_j g \pmod{p} \quad (\text{for } j = 1, 2), \\ \gcd(A_1, A_2) = g, \\ A_1B_2 - A_2B_1 = p, \\ 2^{k_1}A_1 \leq p, \quad 2^{k_2}A_2 \leq x, \\ B_1 \leq 0 < B_2. \end{aligned} \tag{1}$$

When the loop exits, a modular multiplication by a table look-up removes powers of 2 from the output. We can bound the subscript $k_1 + k_2$ by

$$2^{k_1+k_2} \leq (2^{k_1}A_1)(2^{k_2}A_2) \leq px.$$

We will have $A_1 = A_2 = \gcd(A_1, A_2) = g$. If $A_2 > 1$ then we report an error to the caller. Otherwise $g = 1$. The output $z = B_2 \cdot (2^{-k_1-k_2})$ satisfies

$$z = zg \equiv B_2 \cdot (2^{-k_1-k_2})g \equiv (A_2 2^{k_1+k_2}y)2^{-k_1-k_2} \equiv A_2 y = y \pmod{p}.$$

A shift reduction always starts with at least one of A_1 and A_2 being odd, by (1). We do not know which of these might be even, but can examine both, in a SIMD fashion. If we pick t_1 and t_2 as large as possible during a shift reduction, then the new A_1 and A_2 will both be odd. In that case the next subtraction and shift reductions will reduce $A_1 + A_2$ by at least a factor of 2. The trailing zero bit count of a positive integer A is the population count of $\overline{A} \wedge (A-1)$. The SPU's population count instruction acts only on 8-bit data, so our t_1 and t_2 may not be maximal.

The values of A_1 and A_2 are bounded by p and x, respectively. The invariant $p = A_1B_2 - A_2B_1 \geq B_2 - B_1$ bounds B_1 and B_2. For $n = 4$ these fit in 128 bits.

Within the subtraction reduction, the four 128-bit differences $A_1 - A_2$, $B_1 - B_2$, $A_2 - A_1$, and $B_2 - B_1$ are evaluated in parallel. We exit the loop if neither $A_1 - A_2$ nor $A_2 - A_1$ needs a borrow. Otherwise we update $[A_1, B_1, A_2, B_2]$ appropriately. Subtracting 1 from each element of the borrow vector gives masks of -1 or 0 depending on the sign of $A_1 - A_2$ or $A_2 - A_1$. A shuffle of these masks builds a selector which determines which parts of $[A_1, B_1, A_2, B_2]$ are updated.

The final multiplication with 2^{-k} is done by first looking up this value in a table and next computing the modular multiplication as outlined in Prop. 1. Hence, the modular inversion implementation takes as input an integer x and outputs $z \equiv \frac{1}{x} \bmod p$ with $0 \leq x, z < \widetilde{p}$.

Computation of a single modular inverse takes fewer than 5000 clock cycles on average. With $N = 400$ Pollard rho processes running in parallel, it boils down to about 12 clock cycles per Pollard rho process (plus the time required for the additional modular multiplications required per process in order to be able to use simultaneous inversion).

A.4 Incorporation into the Pollard rho iteration

Given elliptic curve group elements P and Q, we wish to calculate an integer m such that $Q = mP$. Since we use 16-adding walks (cf. [43]) we fix 16 group elements R_j, $0 \leq j < 16$,

calculated as $R_j = \tilde{c}_j P + \tilde{d}_j Q$ for random integers $\tilde{c}_j, \tilde{d}_j$ and share those among all Pollard rho processes that will be participating in the calculation, i.e., across all streams on the same SPU, but also among all SPUs, and among all participating PS3s and any other devices. Furthermore, we fix and share an easy-to-calculate function h from the elliptic curve group more or less uniformly to $\{0, 1, 2, \ldots, 15\}$.

For each Pollard rho process integers c_0 and d_0 are randomly selected, and the startpoint $X_0 = c_0 P + d_0 Q$ is calculated. Using elliptic curve point addition (using doubling as well for the calculation of the R_j and all X_0's, but *never* using doubling during the iteration, cf. below), while sharing the inversion cost among $N = 400$ processes on the same SPU, the new point is then calculated as $X_{i+1} = X_i + R_{h(X_i)} = c_{i+1} P + d_{i+1} Q$ for $i = 0, 1, 2, \ldots$ until a distinguished point is found (cf. [44]). At the cost of 16 counters of 32 bits each per process, updating the values c_{i+1}, d_{i+1} can be postponed until a distinguished point is found. Both the calculation of h and the check for the distinguished point property require unique representation of the group element, whereas we use a redundant representation modulo $\tilde{p}$. This is relatively easy to fix by calculating the unique partial Montgomery reduction $x \cdot 2^{-16} \bmod p$ of the x-coordinate of the group element in question, since it will be unique in $\{0, 1, \ldots, p-1\}$. Identical distinguished points with, most likely, different constants c and d may lead to the desired solution to $Q = mP$.

As noted above, several things can go wrong: we may have dropped off the curve because we should have used curve doubling (in the unlikely case that $X_i = R_{h(X_i)}$, or in the unlikely case of incorrect reduction modulo $\tilde{p}$, cf. Prop. 1), or a wrong point may by accident again have landed on the curve, and have nonsensical c_i, d_i values. Just as the correct iterations, these wrong points will after a while end up as distinguished points. Thus, whenever a point is distinguished, we check that it indeed lies on the curve and that the equation $x_i = c_i P + d_i Q$ holds for the alleged c_i and d_i. Only correct distinguished points are collected. If we hit upon a process that has gone off-track, all 400 concurrent processes on that SPU are terminated and restarted, each with a fresh startpoint. This type of error-acceptance leads to enormous time-savings and code-size reduction at negligible cost: we have not found even a single incorrect distinguished point yet.

Enforcing uniqueness using the partial Montgomery reduction takes on average 24 clock cycles per iteration per process. The other issues mentioned above (retrieving the correct R_j's, shuffling them into the right places, distinguished point checking, branching overhead) contribute another 68 clock cycles, on average.

A.5 Timings

Table 1 combines the average clock cycle requirements per Pollard rho process. The total expected number of iterations to solve a discrete logarithm problem in the elliptic curve group over the 112-bit prime field is $\sqrt{\frac{\pi \cdot q}{2}} \approx 8.4 \cdot 10^{16}$, where q is the prime group order and where we do not use the negation map. Disregarding the possibility of tag-tracing, we need 456 clock cycles per iteration per SPU. Since an SPU runs at 3.2GHz and 6 SPUs are available per PS3, we expect about 60 PS3 years to complete the calculation. A moderate speed-up can be expected if one of the two improvements is used.

Operation	Average #cycles per operation	Quantity per iteration	Average #cycles per iteration
Modular multiplication	54	6	322
Modular subtraction	5	6	30
Modular inversion	4941	$\frac{1}{400}$	12
Montgomery reduction	24	1	24
Miscellaneous	68	1	68
Total			456

Table 1. *Average clock cycle count for the operations during a Pollard rho iteration on one SPU.*

With a 24-bit distinguishing property we expect $5 \cdot 10^9$ distinguished points. Storing each distinguished point along with its c and d value requires $4 \cdot 112$ bits, i.e., 56 bytes, for a total of about $56 \cdot 5 \cdot 10^9$ bytes, i.e., 260 gigabytes.

Given the breakdown of the 54 cycle count for modular multiplication into (regular schoolbook) multiplication and (fast, due to Prop. 1) reduction, we expect that for generic 112-bit moduli only the 322 in Table 1 needs to be changed, namely to approximately 420. As a result the overall cycle count would grow by about 20%.

More precise extrapolation of the iteration cycle count for a generic 112-bit modulus to a generic 160-bit one than the rough "$\left(\frac{160}{112}\right)^2 \approx 2$" used in the PS3-part of Section 3, results in $N = 320$ and an overall iteration cycle count less than 900. The 160-bit ECC PS3-estimate from Section 3 is therefore on the high side (cf. also last paragraph of Section A.1).