Offline Submission with RSA Time-Lock Puzzles

Yves Igor Jerschow Martin Mauve

Institute of Computer Science, Heinrich Heine University, Düsseldorf, Germany
{jerschow, mauve}@cs.uni-duesseldorf.de

***Abstract*—We introduce a non-interactive RSA time-lock puzzle scheme whose level of difficulty can be arbitrarily chosen by artificially enlarging the public exponent. Solving a puzzle for a message m means for Bob to encrypt m with Alice's public puzzle key by repeated modular squaring. The number of squarings to perform determines the puzzle complexity. This puzzle is non-parallelizable. Thus, the solution time cannot be shortened significantly by employing many machines and it varies only slightly across modern CPUs. Alice can quickly verify the puzzle solution by decrypting the ciphertext with a regular private key operation. Our main contribution is an offline submission protocol which enables an author being currently offline to commit to his document before the deadline by continuously solving an RSA puzzle based on that document. When regaining Internet connectivity, he submits his document along with the puzzle solution which is a proof for the timely completion of the document. We have implemented a platform-independent tool performing all parts of our offline submission protocol: puzzle benchmark, issuing a time-lock RSA certificate, solving a puzzle and finally verifying the solution for a submitted document. Two other applications we propose for RSA time-lock puzzles are trial certificates from a well-known CA and a CEO disclosing the signing private key to his deputy.**

I. Introduction

Due to the evolution of the Internet, online submission of documents like conference papers, homework assignments, applications or claims has become very popular. Many institutions even establish paperless electronic submissions as the only submission mode, since it significantly reduces their processing costs. Each call for submission has, of course, its deadline and each document received past the time limit has to be rejected by the institution for fairness reasons. However, there may be situations where the document is completed in time, but cannot be submitted by the author before the expiration of the deadline because of technical issues. One possible reason may be a broken network connection in all its flavors, e. g., the access network—be it ADSL, UMTS, WiFi or dialup—becoming temporarily unavailable, an ISP failure or a DNS resolution problem. The submission server itself may also become temporarily unreachable due to a crash or a Denial-of-Service (DoS) attack. Finally, it is also conceivable that by the time of the deadline the author stays in a remote region without Internet access and therefore cannot submit the document in time. Today, in all these scenarios the author just has bad luck and there is nothing he can do about it, since the institution accepting the document is usually not able to verify and thus to consider any mitigating circumstances.

In this paper, we propose a new cryptographic protocol inspired by Rivest's *time-lock puzzles* [1]. It enables an author to commit to a document in an offline manner before the deadline and to submit it at some time past the deadline when being online again. The main idea is to let the author solve a modular exponentiation puzzle involving an arbitrary large number of *non-parallelizable* modular squaring operations. We construct the puzzle from the document's cryptographic hash value. The number of puzzle operations is determined by the time period between the deadline and the point in time where the author regains connectivity to the submission server. Each puzzle operation has a time value of some nanoseconds assigned by the institution managing the submission process and is dictated by current CPU speeds. By submitting his document along with the appropriate puzzle solution the author can prove to the institution that the document has actually been completed at some time in the past before the deadline.

We introduce a time-lock RSA puzzle scheme for delayed encryption and signature verification. The basis of our offline submission protocol is a delayed RSA encryption of the document to be submitted using the institution's public key. Having received the delayed submission, the institution verifies the puzzle solution and the assigned level of difficulty by performing an RSA decryption with its private key. Running the offline submission protocol requires the author to hold a computer with a reasonably up-to-date processor and to continuously solve the puzzle from the expiration of the deadline until the actual online submission. Owners of older hardware can compensate by completing the document and beginning to solve the puzzle at some point before the actual deadline—the earlier the better. We show that in combination with the non-parallelizability feature the difference in puzzle processing speed between recent off-the-shelf computers usually does not exceed factor 1.5.

We have implemented a platform-independent tool which performs all parts of our offline submission protocol: puzzle benchmark, issuing a time-lock RSA certificate, solving a puzzle and finally verifying the solution for a submitted document. The tool is available for free download including the sources and can be instantly used by the two parties—institution and author—to enable a delayed submission for an online submission system.

The remainder of this paper is organized as follows. In the next section, we discuss existing approaches on time-lock

cryptography. Section III introduces our RSA time-lock puzzle scheme. In Section IV we describe how to construct an offline submission protocol on that basis. Section V presents the implementation of our offline submission tool and evaluates its performance. Finally, we conclude the paper with a summary in Section VI.

II. Related Work

A. Time-Lock Puzzles

Time-lock puzzles have been introduced by Rivest et al. [1] to encrypt messages which can be decrypted by others only after a pre-determined amount of time has passed. Possible applications proposed for timed-release cryptography are: sealing bids in an auction which cannot be opened prior the end of the bidding period, releasing documents like diaries in the future, scheduling electronic payments, or implementing a key-escrow scheme where the government can get a secret key after a fixed period. Non-parallelizability of the underlying repeated squaring operation makes up the key feature of time-lock puzzles—the solver cannot speed-up the computation by engaging multiple CPU cores or machines. In contrast, hash-based *client puzzles* proposed by Juels and Brainard [2] as a countermeasure against DoS attacks are fully parallelizable. The task of reversing a one-way hash function by brute force can be easily distributed across many machines.

Rivest's time-lock puzzle is in a way related to his RSA cryptosystem and works as follows: To encrypt a message m for a period of T seconds Alice

- generates at random two large primes p and q.
- computes the modulus $n = p\,q$ and Euler's totient function $\varphi(n) = (p-1)\,(q-1)$.
- determines the number of squaring operations modulo n per second, denoted by S, that can be performed by the solver Bob, and computes $t = T \cdot S$.
- encrypts m with a symmetric cipher using the key K.
- picks a random a, $1 < a < n$, and encrypts K as

$$C_K = K + a^{2^t} mod\ n. \tag{1}$$

  To make the exponentiation efficient, Alice reduces the exponent modulo $\varphi(n)$ by computing

$$r = 2^t\ mod\ \varphi(n) \tag{2}$$

  and obtains $a^{2^t} mod\ n$ from $a^r\ mod\ n$.
- outputs the time-lock puzzle (n, a, t, C_K).

To reveal K from C_K, Bob needs to compute $a^{2^t} mod\ n$ and in contrast to Alice cannot take the shortcut via $\varphi(n)$, since determining $\varphi(n)$ is provably as hard as factoring n. Instead, Bob must do the computation step by step by repeatedly performing modular squarings—altogether t times which takes T seconds. This is assumed to be an intrinsically sequential process since each next step requires the intermediate result from the previous one. Parallelization of the squaring operation itself cannot achieve a significant speedup either. Each squaring requires only trivial computational resources and any non-trivial scale of parallelization inside the squaring operation would be likely penalized by communication overhead among the processors [3].

In complexity theory, the class P contains all decision problems that can be solved by a deterministic Turing machine in polynomial time. $\mathsf{NC} \subseteq \mathsf{P}$ represents the class of problems that can be efficiently solved by a parallel computer. It is still an open question whether modular exponentiation is P-complete, i. e., not in NC [4], [5]. Likewise, it is unknown if factoring is really not in P. Hence, the security of time-lock puzzles is based on these two unproven assumptions which are known to be hard problems for many years.

A comprehensive survey on efficient algorithms for modular exponentiation can be found in [6] and [7]. The most important algorithms beside the basic binary exponentiation are the k-ary method, the sliding-window method, and addition chains. However, when dealing with a power-of-two exponent as is the case with time-lock puzzles, repeated squaring—a special case of the binary exponentiation—constitutes the most efficient technique. To compute $a^x\ mod\ n$ with $x = 2^t$ it takes t modular squarings and no additional multiplications while $\lfloor log\ x \rfloor$ is the lower bound for the number of multiplications to perform a single exponentiation in a general group. To accelerate the modular multiplication, especially when being performed repeatedly during modular exponentiation, Montgomery proposed to use an alternative representation of integers modulo n, called the Montgomery reduction [8]. It allows to carry out the modular multiplication without performing the classical modular reduction step. Instead, the more efficient Montgomery reduction is applied.

While the costs of solving the time-lock puzzle in an optimal way are well known, the release time will vary depending on the speed of the recipient's processor and is somewhat coarse-grained. However, Rivest argues that the speeds of hardware available to consumers differ only by a small constant factor and even the power of high-end hardware available to companies is usually within the same order of magnitude due to non-parallelizability of the problem. We agree on this rationale and further investigate it by comparing the puzzle solution times on different off-the-shelf machines. Our offline submission protocol tolerates authors with slower machines if they start to solve their puzzle at some time before the deadline.

B. More Timed-Release Cryptography

In [3] Mao developed a zero-knowledge protocol which enables Alice to prove to Bob that a *timed encryption* or a *timed signature* based on time-lock puzzles can be actually unlocked by performing t modular squarings. Boneh and Naor [9] introduced a verifiable timed commitment scheme extending the standard notion of commitments. It adds a potential forced opening phase which permits the receiver to recover with some effort the committed value without the help of the committer. Like in time-lock puzzles, the recovery rests upon repeated squaring. Possible applications for timed commitments are contract signing, honesty-preserving auctions, and concurrent zero-knowledge. Building on the work of Boneh and Naor,

Garay and Jakobsson proposed a timed release scheme for standard digital signatures—RSA, Schnorr and DSA [10].

A different approach to timed-release cryptography that does not require the receiver to solve a puzzle and provides fine-grained timing is presented by Blake and Chan [11]. They assume a trusted time server which periodically broadcasts signed time-bound key updates I_t to the users. The time server does not need to interact with either the sender or the receiver and is therefore passive. At release time t the receiver can decrypt his message by means of I_t. This scheme is based upon a bilinear pairing. Cathalo et al. [12] improved it by introducing a new stringent security model and strengthening the anonymity of receivers. Other contributions to timed-release cryptography using trusted time servers are, e. g., [13]–[15]. In contrast, we pursue an offline approach and cannot rely on or even assume the presence of a trusted time server.

III. RSA Time-Lock Puzzle Scheme

A. Key Generation

We incorporate the time-lock puzzle mechanism into the default RSA public-key cryptosystem and make the puzzle non-interactive. Everyone who knows Alice's public puzzle key can solve a puzzle by encrypting an arbitrarily chosen message m. The puzzle complexity is determined by the size of Alice's public key. Alice constructs her RSA puzzle key pair with the artificially enlarged public key by performing the following steps:

1) Generate at random two large primes p and q of equal bit-length (e. g., 1024 bits).
2) Compute the modulus $n = p\,q$ and Euler's totient function $\varphi(n) = (p-1)\,(q-1)$.
3) Randomly choose a private exponent d, $1 < d < \varphi(n)$, such that $gcd(d, \varphi(n)) = 1$ and determine its multiplicative inverse modulo $\varphi(n)$: $e = d^{-1} \ mod \ \varphi(n)$.
4) Choose the puzzle difficulty t which is the number of modular squarings Bob has to perform to solve the puzzle, i. e., to carry out the public key operation. Suppose that a high-performance reference machine can do S squarings modulo n per second and a public key operation shall take T seconds, then $t = T \cdot S$.
5) Compute the remainder
$$r = 2^t \ mod \ \varphi(n) \tag{3}$$
and the public exponent
$$\tilde{e} = 2^t + \varphi(n) - r + e. \tag{4}$$
$z = \varphi(n) - r + e$ denotes the lower bits of $\tilde{e}$ which are preceded by a long sequence of `0`-bits and finally the leading `1`-bit at position t.
6) $(n, \tilde{e})$ is the public and (n, d) the private key. Since $\tilde{e}$ is an extremely large number with lots of `0`-bits after the leading `1`-bit, the public key can be efficiently represented by storing the triple (n, t, z). In binary, z is at most twice as long as n.

The inflated public exponent $\tilde{e}$ is constructed by adding a large multiple of $\varphi(n)$ to the regular exponent e. It holds that $m^e \equiv m^{\tilde{e}} \ (mod \ n)$ for all $m \in \mathbb{Z}_n$, since $e \equiv \tilde{e} \ (mod \ \varphi(n))$ and n is a product of distinct primes. $\tilde{e}$ has been chosen to be the smallest appropriate exponent which is larger than 2^t. The time to perform the modular reduction of 2^t in step 5 depends, of course, on the puzzle difficulty t. However, even when creating a puzzle with a solution time in the order of several days, step 5 will take only a few minutes.

B. Public and Private Key Operation

Solving a puzzle for a context m, $0 < m < n$, chosen by the solver Bob means to carry out the public key operation by encrypting m with Alice's public key $(n, \tilde{e})$ in the usual manner, i. e., to compute the ciphertext
$$c = m^{\tilde{e}} \ mod \ n. \tag{5}$$
Due to the special structure of $\tilde{e}$, the fastest way to perform this giant modular exponentiation is to solve the actual puzzle
$$\alpha = m^{2^t} \ mod \ n \tag{6}$$
in T seconds by repeated squaring and to quickly do the regular-sized modular exponentiation
$$\beta = m^z \ mod \ n \tag{7}$$
which yields
$$c = \alpha \cdot \beta \ mod \ n. \tag{8}$$
Bob submits the pair (m, c), i. e., the context and the corresponding puzzle solution, to Alice. She verifies the solution by applying her private key (n, d) in the usual manner to decrypt the ciphertext and to compare the result with m:
$$c^d \ mod \ n \stackrel{?}{=} m. \tag{9}$$
Since d is of regular size, this operation takes just a few milliseconds. If the verification succeeds, Alice is convinced that Bob has spent about T seconds to solve the puzzle (or even longer, if his computer ist not as fast as Alice's high-end reference machine).

C. Security Analysis

The security of our RSA puzzle scheme can be reduced to the security of Rivest's puzzle construction. It must be impossible for Bob to compute c without performing the t modular squarings in (6). Determining $\varphi(n)$ in order to reduce $\tilde{e}$ to e is provably as hard as factoring n and therefore is not an option. Bob knows 2^t, $\tilde{e} = 2^t + z$ and $z = \varphi(n) - r + e$ respectively, but has no information about the individual summands $\varphi(n)$, $-r$, and e. With regard to $\varphi(n)$ and $r = 2^t \ mod \ \varphi(n)$, the case is the same as in Rivest's scheme. Being the modular inverse of the randomly generated number d, e is completely random as well and therefore is not correlated with either $\varphi(n)$ or r. Thus, we cannot identify $\varphi(n)$ or r from z. The only possibility remaining is to determine e from z if some information on the relationship between $\varphi(n)$ and r is known. Suppose Bob can easily find the difference $\varphi(n) - r$, then Rivest's scheme

would be broken as well. In this case Bob would be able to compute

$$y = a^{\varphi(n)-r} \ mod \ n = a^{-r} \ mod \ n = (a^r)^{-1} \ mod \ n \quad (10)$$

and to determine the puzzle solution $a^{2^t} mod \ n$ by inverting y modulo n. For the very unlikely case that y is not invertible, $gcd(y, n) = p$ or $gcd(y, n) = q$ and we have factored n.

It is crucially important that after publishing $\tilde{e}$ Alice never reveals for the same key pair another exponent $\hat{e}$, e. g., a smaller one to make the puzzle easier. Otherwise the modulus n could be factored quite quickly. $\delta = \tilde{e} - \hat{e}$ is a multiple of $\varphi(n)$ and there exists an efficient randomized algorithm which allows to factor n if a multiple of $\varphi(n)$ is given [16]. Though the algorithm requires to perform at least one modular exponentiation with an exponent in the order of δ, i. e., takes about as long as solving one instance of the puzzle, knowing the factorization of n enables to solve all future puzzles instantly.

D. Delayed Encryption and Signature Verification

Our RSA time-lock scheme can be used not only to solve puzzles, but also to delay the regular RSA encryption and signature verification process. Using the public exponent $\tilde{e}$ instead of e the public key operation will take about T seconds where T can be chosen arbitrarily. What is this good for? We propose two possible applications.

The first one is a well-known certificate authority (CA) which decides to provide its services for advertising purposes free of charge or a for very low fee, if the certificate holders accept a restriction on the computational speed of their public key. Companies and large organizations usually do not bother about the certification fee and buy a full-fledged certificate. Thus, the primary target group would be individuals and small societies who often cannot afford to pay the regular fee. Instead of limiting the validity of a trial certificate to some days which makes it actually useless, the CA would accept only artificially enlarged public exponents for long-term certification within the promotion. It could prescribe to provide a public exponent of the form $2^t + z$ where t is chosen as large as to perform the public key operation in not less than T seconds. Reasonable values for T may be, e. g., 60 seconds for a free and 10 seconds for a low-fee certificate. Such an overhead when encrypting a message for the certificate holder or verifying his signature would not constitute a serious limitation for parties with whom individuals or small societies usually communicate. The proposed marketing strategy would make the CA even more popular and leverage the deployment of public key cryptography.

The second application focuses on delayed signature verification in the context of contract signing. In a company only very few persons should be authorized signatories, i. e., possess the company's private key enabling them to sign arbitrary contracts on behalf of the company. Besides the CEO, there may be only one deputy who has access to the private key and even he may not enjoy the CEO's full confidence. The CEO will be keen on to restrict the deputy's signing capability but must pay attention not to compromise the company's capacity to act in case of his sudden absence or illness. Our approach to this dilemma is for the CEO to generate two key pairs and to certify for his company two public keys. The first and regular public key is of normal size while the second one is an artificially enlarged puzzle key (n, t, z) and takes, e. g., $T = 48\ h$ per operation. The private key corresponding to the regular public key would be known solely to the CEO, while the second private key is disclosed to the deputy. Computing a signature is an easy task with both private keys. However, only a signature created with the CEO's private key can be efficiently verified. Under normal circumstances all current contracts are signed by the CEO and the other party can immediately check the signature. Concluding an agreement with the deputy is not attractive due to the extremely time-consuming signature verification. But in case that the CEO is temporarily not available, the only way to stay in business is for the deputy to sign the pending contract and for the other party to be patient while validating the signature. Except for this inconvenience, the other party receives a full-fledged signature which, if necessary, can be presented in court. It will take the court once again time T to check the signature, but this is not an issue. As soon as the CEO is available, he may resign the contract with his private key yielding a quickly verifiable signature. Holding a private key whose genuineness cannot be easily validated, the deputy is much less vulnerable to attempts to rapidly extort the key under threat of violence than the CEO. Under the condition that the deputy does not know $\varphi(n)$, which he does not need to know to generate signatures, the hijackers would have to wait for time T to test whether the revealed private key is actually genuine. Instead, in case of sharing the regular private key, both the CEO and his deputy would be worthwile targets.

E. Other Applications for RSA Time-Lock Puzzles

Generally speaking, the solution of an RSA time-lock puzzle constitutes a non-interactive and non-parallelizable proof of work for an arbitrarily chosen context m that took (at least) time T. Beyond the offline submission that we present in the next section, one could make use of RSA puzzles to enable an ordinary citizen to get an appointment with a high-ranking politician, e. g., a mayor or a minister, and to discuss a crucial concern m. By solving a long-term puzzle for m the citizen demonstrates that he really has a strong intention and deserves to be listened to. This increases his chances for getting a time-slot for the concern m—and only for it.

F. Small Private Exponent

To speed up the private key operation, the private exponent d can be chosen considerably smaller than the modulus n. Boneh and Durfeecite [17] showed that as long as $d < n^{0.292}$, one can break RSA by recovering the private exponent from the public key. However, this attack on small private exponents is only feasible if the public exponent $e < n^{1.875}$. Hence, since our RSA puzzle scheme relies on an extremely large public exponent, Boneh's attack does not apply here. Of course,

d must be chosen large enough that it cannot be guessed by brute force. A minimal size in the same order of magnitude as symmetric keys seems to be appropriate, e. g., 128–192 bits.

IV. Offline Submission Protocol

Based on the RSA time-lock puzzle scheme, we propose now an offline submission protocol which enables an author currently being offline to commit to its ready-made document before the deadline and to submit it at some time past the deadline upon regaining connectivity. The goal is to convince the accepting institution of the timely completion of the document by means of a successfully solved RSA puzzle.

A. Basic Design

The institution generates an RSA puzzle key pair where the public key operation takes time T on a reference machine being equipped with a state-of-the-art high-end processor. It can perform S modular squarings per second and should be one of the fastest systems available on the market to end users. Setting the bar high is important to ensure that nobody can gain a time advantage over other authors who submit in time. The institution publishes the public puzzle key (n, t, z) in the usual fashion, e. g., by requesting a certificate from a trusted CA and making it available on its website and in public key directories. An author intending to submit a document obtains the puzzle certificate in advance—just in case he has no Internet connection to the submission server when the deadline approaches. Many different scenarios are conceivable, ranging from hardware or ISP failure, a cable break, a DoS attack on the submission server to a location-dependent unavailability of Internet access in a remote region.

Should this be the case, the author begins to solve an RSA puzzle for his document. Note that electricity to run the computer is usually available even in an adverse environment. He applies a cryptographic hash function (e. g., SHA-1 or RIPEMD-160) to his document producing a digest which serves as input m for the puzzle. If his computer is as fast as the reference machine, he computes the solution $c = m^{\tilde{e}} \ mod \ n$ in time T. Assuming that at that time the Internet connection to the server is available again, the author finally submits its document along with the puzzle solution c. Now the institution verifies the solution by decrypting c with its private key and matching the result against the document's hash value: $c^d \ mod \ n \stackrel{?}{=} m$. If the validation succeeds, the institution is convinced that the author has finalized his document at least T seconds ago. Is this point in time before the deadline, the submission can be predated and accepted. It is up to the institution to specify a maximum submission delay beyond which no documents can be considered any more due to the closure of the review process.

In case that the author holds a slower processor than the reference machine, he can compensate for this handicap by beginning to solve the puzzle at some point before the deadline—ideally, just after the finalization of the document. Let S' denote the number of modular squarings that the author's machine can perform per second, then he must start solving a puzzle designed for T seconds at least $(\frac{S}{S'} - 1)T$ seconds before the deadline to succeed.

B. Building a Puzzle Chain

In practice, the author cannot predict exactly when he regains connectivity to the submission server. Solving a single but very complex puzzle which probably takes more time than the period without Internet access lasts would be suboptimal, especially for owners of older hardware. Therefore we propose for the institution to issue several public puzzle keys with different levels of difficulty, e. g., one for 12 hours, for 4 hours, for 1 hour, and one for 10 minutes. The author can estimate the anticipated offline time and begins to solve the most suitable puzzle. If he is still offline after having solved the first puzzle, he continues to solve puzzles by building up a *puzzle chain*: The solution c_1 of the first puzzle becomes the input m_2 of the second, usually shorter lasting puzzle. The author continues to chain up his puzzle solutions according to this scheme until he finally regains connectivity to the server after k puzzle steps. Then he can submit his document along with the k chain links $c_1, ..., c_k$. Each solution should bear a label stating the public key used. The institution now validates the chain by verifying each puzzle solution: $c_i^{d_i} \ mod \ n_i \stackrel{?}{=} m_i$ for $1 \leq i \leq k$ where $m_1 = m$ and $m_i = c_{i-1}$ for $i > 1$. Note that this task can be performed in parallel. Summing up the times T_i assigned to the utilized public keys yields the total time by which the submission is predated.

C. Alternative Approach

Another approach for solving the puzzle only as long as necessary is for the author to choose the large exponent for the computation by himself. He could simply compute $c = m^{2^t} \ mod \ n$ by repeated squaring for a t which is as large as he actually needs, i. e., the final t would be the number of modular squarings performed until the Internet connection becomes available again. This approach would ignore the RSA property of the original puzzle construction and require only the modulus n along with the speed indication S from the reference machine. The institution would need to compute $r = 2^t \ mod \ \varphi(n)$ first prior to verifying $m^r \ mod \ n \stackrel{?}{=} c$. A drawback of this scheme is the relatively expensive modular reduction of 2^t which must be rerun for each submitted puzzle instead of performing it only once during the key generation. Moreover, in the modular exponentiation $m^r \ mod \ n$ the exponent r is roughly the same size as n, while in the RSA puzzle scheme a smaller private exponent d can be chosen, see Section III-F. Verifying a short chain of RSA puzzles is therefore several orders of magnitudes faster.

V. Implementation and Evaluation

A. The OSRTLP Tool

We have implemented a platform-independent tool in C++, called *OSRTLP*[1], which performs all parts of our offline submission protocol. It is available for free download including

[1]This is the acronym for *Offline Submission with RSA Time-Lock Puzzles*.

TABLE I
PERFORMANCE COMPARISON OF THE MODULAR SQUARING OPERATION ON DIFFERENT PLATFORMS.

platform	*CPU release date & price*	*S: modular squarings / sec*		
		1024 bits	2048 bits	4096 bits
Intel Core 2 Duo E6400 2.13 GHz Linux 2.6.31 64-bit	07 / 2006 183 $	941 320	261 750	71 340
Intel Core 2 Duo E6750 2.66 GHz Windows 7 32-bit	07 / 2007 183 $	290 420	80 790	21 520
Windows 7 64-bit		1 161 860	323 410	87 880
Linux 2.6.31 32-bit		328 670	94 340	26 360
Linux 2.6.31 64-bit		1 174 160	324 670	88 590
Intel Core 2 Quad Q9400 2.66 GHz Linux 2.6.31 64-bit	08 / 2008 183 $	1 180 970	326 250	88 810
Intel Core 2 Duo T9900 3.06 GHz Linux 2.6.31 64-bit	04 / 2009 530 $	1 396 290	386 330	104 780
Intel Xeon X3360 2.83 GHz Linux 2.6.31 64-bit	03 / 2008 266 $	1 237 160	346 730	93 940
AMD Athlon II X2 240e 2.80 GHz Linux 2.6.31 64-bit	10 / 2009 77 $	1 092 270	345 080	99 600

the sources (with Visual C++ project, GNU Makefile and precompiled binaries for Windows) [18]. At the beginning, the institution can use OSRTLP for running a puzzle benchmark on a high-end reference machine to determine the number of modular squaring operations S executed per second. Next, it creates an RSA key pair with a public puzzle key taking T seconds per operation. Both the modulus size n and puzzle time T can be chosen arbitrarily. OSRTLP outputs the puzzle's private key and a puzzle certificate in X.509 v3 format containing, besides subject information and public puzzle key, the puzzle time T. It is signed by the institution's CA private key. If necessary, the institution may ask a well-known CA to cross-certify its CA public key. The author utilizes OSRTLP to solve a puzzle for his document by supplying the institution's puzzle certificate. It can be verified by OSRTLP against a trusted CA certificate (or even a chain). At first, OSRTLP performs a short benchmark to inform the user about the time expected to finish the puzzle and indicates the current progress in percent. One can choose between the hash functions SHA-1, SHA-256 and RIPEMD-160. While solving the puzzle, OSRTLP periodically backups the intermediate result to a file and can simply resume the computation in case of a crash. Finally, the institution runs OSRTLP to quickly verify the solution for a submitted document by applying the puzzle's private key.

For the large-integer arithmetic we employ the open source library *MPIR* [19] which is a fork of the well-known *GMP* library from GNU [20]. GMP claims to be faster than any other bignum library by using fast algorithms with highly optimized assembly code. This serves our needs very well since we aim to provide a puzzle solver which cannot be easily outperformed. The institution must have confidence that the author is not able to solve the puzzle quicker than supposed, at least not at an acceptable price. MPIR / GMP implements several state-of-the-art multiplication algorithms, ranging from the base-case schoolbook method to the Karatsuba, Toom-Cook, and FFT algorithms. The choice depends on the bit length. For squaring integers which have the size of a typical RSA modulus, i. e., 1024–4096 bits, MPIR / GMP resorts to the schoolbook and Karatsuba method. The thresholds are platform-dependent. On current CPUs, for integers larger than 1536–1920 bits Karatsuba'a algorithm, running in $\mathcal{O}(N^{1.585})$, outperforms the basecase $\mathcal{O}(N^2)$ method. N denotes the number of machine words (in practice, 32 or 64 bits long) required to represent the integer. For repeated modular squaring we make use of Montgomery reduction instead of performing the classical reduction by dividing. This speeds-up the puzzle solution by a factor of 1.3–2.0, especially for small moduli in the order 1024–2048 bits. The private key operation for puzzle verification is also optimized by performing two exponentiations modulo p and q and afterwards applying the Chinese Remainder Theorem which yields the solution modulo n.

All MPIR / GMP functions operate on integers which are completely stored in memory. However, 2^t is far too large to be held in memory and consists almost only of zeros. To perform the modular reduction $r = 2^t \mod \varphi(n)$ we have therefore modified the library's division routine to efficiently represent the dividend by occupying storage space only in the order of the modulus (i. e., the divisor). The same issue arises when storing the public exponent $\tilde{e} = 2^t + z$ in an X.509 certificate. We address it by encoding $\tilde{e}$ as the odd integer $E = z \cdot 2^{65} + t \cdot 2^1 + 1$ where t is represented as a 64-bit integer. Such a puzzle time-lock certificate can be distinguished from a regular one by a time-lock indication in the subject alternative name extension.

Fast modular multiplication has been also successfully implemented in hardware, especially on FPGAs [21], [22], and for modern GPUs [23], [24]. The FPGA implementations are very competitive and a few years ago they outperformed ordinary software implementations. However, a current comparison [24] shows that nowadays FPGA implementations are about as fast as software implementations on up-to-date CPUs. GPUs also do not surpass CPUs in general, at least not when running a single modular exponentiation as is the case with our puzzle. Moreover, special purpose hardware like FPGAs is quite expensive, so the great majority of authors would not buy it for offline submission.

B. Performance Evaluation

We run OSRTLP in benchmark mode on different platforms to measure the number of modular squaring operations S that each machine can perform per second. Our goal is to compare to what extent the puzzle solution time differs between an up-to-date high-end CPU being a candidate for the reference machine and a processor that was purchased some years ago. We also investigate the difference between 32-bit and 64-bit architectures and the impact of the operating system. We compiled OSRTLP and MPIR 1.3.1 with GCC 4.4.1 on Linux and Visual C++ 2008 SP-1 on Windows. The results for 1024,

TABLE II
COMPUTATION TIME OF $r = 2^t \bmod \varphi(n)$ ON AN INTEL CORE 2 DUO E6750 2.66 GHZ FOR DIFFERENT PUZZLE DIFFICULTIES $t = T \cdot S$ WITH AN INTEL CORE 2 DUO T9900 3.06 GHZ AS REFERENCE MACHINE FOR S.

puzzle time T	*modulus size* n		
	1024 bits	2048 bits	4096 bits
10 min	0.754 sec	0.292 sec	0.132 sec
1 h	4.512 sec	1.738 sec	0.791 sec
12 h	53, 98 sec	20, 91 sec	9.50 sec
24 h	108.0 sec	41, 84 sec	18.98 sec
72 h	324.2 sec	125.7 sec	56.93 sec

2048, and 4096 bit moduli, all averaged over multiple runs, are shown in Table I. To make it easier putting in relation the different CPUs, we state their release date as well as the manufacturer's release price (in 1000-unit quantities).

Evaluating the results, two main observations can be made: First, a 64-bit implementation of OSRTLP outperforms its 32-bit counterpart by a factor of 3.4 – 4.0. Consequently, in the face of the performance achievable on a 64-bit platform, running a 32-bit version of OSRTLP is not an option. Since all desktop CPUs manufactured during the last four years are 64-bit capable and 64-bit operating systems are widely available, this is in fact not an issue. Second, the difference in speed between 64-bit platforms, ranging from a 3.5 years old Core 2 Duo E6400 2.13 GHz, a 2 years old high-performance Xeon X3360 2.83 GHz to a current Core 2 Duo T9900 3.06 GHz costing 530 $ at release time, amounts to no more than factor 1.5. For the majority of users holding an up-to-date computer the gap between the reference CPU and their own CPU will be actually smaller. This result strongly supports our assumption that non-parallelizable puzzles constitute a feasible approach to measure how much time must have elapsed since the beginning of the computation. Another observation is that the choice of the operating system hardly influences the runtime of the puzzle.

The time required for the institution to perform the modular reduction $r = 2^t \bmod \varphi(n)$ when creating the public puzzle key is indicated in Table II. It is proportional to the desired puzzle solution time T. For a long-term puzzle of several days' duration it takes only a few minutes. The larger the modulus n, the faster the computation of r takes since S decreases for increasing n more quickly than the division speed.

VI. CONCLUSION

In this paper we have introduced a non-interactive and non-parallelizable RSA time-lock puzzle scheme. By artificially enlarging the public exponent the time required to encrypt a message can be arbitrarily tuned. Based on RSA time-lock puzzles, we have proposed an offline submission protocol. It enables an author currently being offline to commit to its document before the deadline and to submit it at some time past the deadline upon regaining connectivity. Presenting the correct solution of a puzzle with assigned solution time T proves to the institution that the submitted document has been finalized at least time T ago. We have implemented a platform-independent tool performing all parts of our offline submission protocol and evaluated the variance of the solution time between different platforms. It turned out to be fairly low.

REFERENCES

[1] R. L. Rivest, A. Shamir, and D. A. Wagner, "Time-lock puzzles and timed-release Crypto," Massachusetts Institute of Technology, Cambridge, MA, USA, Tech. Rep., 1996.

[2] A. Juels and J. G. Brainard, "Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks," in *NDSS '99: Proceedings of the Network and Distributed System Security Symposium*, Feb. 1999.

[3] W. Mao, "Timed-Release Cryptography," in *SAC 2001: Proceedings of the 8th Annual International Workshop on Selected Areas in Cryptography*, Aug. 2001, pp. 342–357.

[4] L. Adleman and K. Kompella, "Using Smoothness to Achieve Parallelism," in *STOC '88: Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, 1988, pp. 528–538.

[5] J. P. Sorenson, "A Sublinear-Time Parallel Algorithm for Integer Modular Exponentiation," in *Proceedings of the Conference on the Mathematics of Public-Key Cryptography*, Jun. 1999, pp. 528–538.

[6] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.

[7] D. M. Gordon, "A Survey of Fast Exponentiation Methods," *Journal of Algorithms*, vol. 27, no. 1, pp. 129–146, 1998.

[8] P. L. Montgomery, "Modular Multiplication without Trial Division," *Mathematics of Computation*, vol. 44, pp. 519–521, 1985.

[9] D. Boneh and M. Naor, "Timed Commitments," in *CRYPTO '00: Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, Aug. 2000, pp. 236–254.

[10] J. A. Garay and M. Jakobsson, "Timed Release of Standard Digital Signatures," in *FC 2002: Proceedings of the 6th International Conference on Financial Cryptography*, Mar. 2003, pp. 168–182.

[11] I. F. Blake and A. C.-F. Chan, "Scalable, Server-Passive, User-Anonymous Timed Release Public Key Encryption from Bilinear Pairing," in *ICDS 2005: Proceedings of the 25th International Conference on Distributed Computing Systems*, Jun. 2005, pp. 504–513.

[12] J. Cathalo, B. Libert, and J.-J. Quisquater, "Efficient and Non-interactive Timed-Release Encryption," in *ICICS 2005: Proceedings of the 7th International Conference on Information and Communications Security*, Dec. 2005, pp. 291–303.

[13] G. D. Crescenzo, R. Ostrovsky, and S. Rajagopalan, "Conditional Oblivious Transfer and Timed-Release Encryption," in *EUROCRYPT '99: Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques*, May 1999, pp. 74–89.

[14] Y. Dodis and D. H. Yum, "Time Capsule Signature," in *FC '05: Proceedings of the 9th International Conference on Financial Cryptography and Data Security*, Mar. 2005, pp. 57–71.

[15] K. Chalkias, D. Hristu-Varsakelis, and G. Stephanides, "Improved Anonymous Timed-Release Encryption," in *ESORICS 2007: Proceedings of the 12th European Symposium On Research In Computer Security*, Sep. 2007, pp. 311–326.

[16] E. Kranakis, *Primality and Cryptography*. John Wiley & Sons, Inc., 1986.

[17] D. Boneh and G. Durfee, "Cryptanalysis of RSA with Private Key d Less than $N^{0.292}$," *IEEE Transactions on Information Theory*, vol. 46, no. 4, pp. 1339–1349, 2000.

[18] Y. I. Jerschow, "The OSRTLP Tool: Offline Submission with RSA Time-Lock Puzzles," http://www.cn.uni-duesseldorf.de/projects/OSRTLP.

[19] "MPIR: Multiple Precision Integers and Rationals," http://www.mpir.org.

[20] "GMP: GNU Multiple Precision Arithmetic Library," http://gmplib.org.

[21] M. M. Ciaran McIvor, J. McCanny, A. Daly, and W. Marnane, "Fast Montgomery Modular Multiplication and RSA Cryptographic Processor Architectures," in *Proceedings of the 37th Asilomar Conference on Signals, Systems, and Computers*, Nov. 2003, pp. 379–384.

[22] D. Suzuki, "How to Maximize the Potential of FPGA Resources for Modular Exponentiation," in *CHES '07: Proceedings of the 9th International Workshop on Cryptographic Hardware and Embedded Systems*, Sep. 2007, pp. 272–288.

[23] S. Fleissner, "GPU-Accelerated Montgomery Exponentiation," in *ICCS '07: Proceedings of the 7th International Conference on Computational Science*, May 2007, pp. 213–220.

[24] R. Szerwinski and T. Güneysu, "Exploiting the Power of GPUs for Asymmetric Cryptography," in *CHES '08: Proceedings of the 10th International Workshop on Cryptographic Hardware and Embedded Systems*, Aug. 2008, pp. 79–99.