Low-Cost Client Puzzles based on Modular Exponentiation

Ghassan O. Karame and Srdjan Čapkun
ETH Zurich, Switzerland
karameg@inf.ethz.ch, capkuns@inf.ethz.ch

Abstract. Client puzzles have been proposed as a useful mechanism for mitigating Denial of Service attacks on network protocols. While several puzzles have been proposed in recent years, most existing non-parallelizable puzzles are based on modular exponentiations. The main drawback of these puzzles is in the high cost that they incur on the puzzle generator (the verifier). In this paper, we propose cryptographic puzzles based on modular exponentiation that reduce this overhead. Our constructions are based on a reasonable intractability assumption in RSA: essentially the difficulty of computing a small private exponent when the public key is larger by several orders of magnitude than the semi-prime modulus. We also discuss puzzle constructions based on CRT-RSA [11]. Given a semi-prime modulus N, the costs incurred on the verifier in our puzzle are decreased by a factor of $\frac{|N|}{k}$ when compared to existing modular exponentiation puzzles, where k is a security parameter. We further show how our puzzle can be integrated in a number of protocols, including those used for the remote verification of computing performance of devices and for the protection against Denial of Service attacks. We validate the performance of our puzzle on PlanetLab nodes.

Key words: Client Puzzles, Outsourcing of Modular Exponentiation, DoS Attacks, Secure Verification of Computing Performance.

1 Introduction

Client Puzzles are tightly coupled with Proof of Work systems in which a client (prover) needs to demonstrate to a puzzle generator (verifier) that it has expended a certain level of computational effort in a specified interval of time. Client puzzles found their application in a number of domains, but their main applications concerned their use in the protection against Denial of Service (DoS) attacks [43, 45, 49] and in the verification of computing performance [13, 44].

To be useful in practice, client puzzles have to satisfy several criteria: namely, they need to be inexpensive to construct and verify, and in many applications should be non-parallelizable. Non-parallelizability of puzzles is an especially important property since clients can involve other processors at their disposal e.g., to inflate their problem-solving performance claim.

A number of puzzles have been proposed [45], but these proposals are either efficient and parallelizable [24, 49] or non-parallelizable and inefficient (typically

in result verification) [13, 43, 44]. Non-parallelizable puzzles are mainly based on modular exponentiation (e.g., [43]); in these puzzles, the verifier has to perform $O(\log(N))$ modular multiplications to construct a puzzle instance and verify its solution. This high cost hindered the large-scale deployment of puzzles based on modular exponentiation in today's online applications [45].

In this paper, we propose puzzles based on modular exponentiation that reduce the cost incurred on the puzzle generator in existing modular exponentiation puzzles. Our constructions are based on a reasonable intractability assumption in RSA: informally, this assumption states that it is computationally intractable to compute a small private exponent d when the public exponent e is larger by several orders of magnitude than the modulus N. It is well known that RSA is insecure when the private exponent is small and the public key $e < N^{1.875}$ [10, 50]. However, when $e \geq N^2$, RSA is considered to be secure [10, 11, 50]. Defeating this assumption would essentially imply a further restriction in the RSA problem, that has not been reported to date. Note that when e is large, the cost of encryption and/or signature verification in RSA is prohibitively high, which explains why this class of RSA keys is not widely used. To the best of our knowledge, this is the first work that leverages on this class of RSA keys to construct low-cost modular exponentiation puzzles. Where appropriate, we also discuss puzzle constructions based on CRT-RSA [11].

Based on this intractability assumption, we show that the costs incurred on the generator of modular exponentiation puzzles can be considerably reduced for any exponent of choice (i.e., for any puzzle difficulty). More specifically, we provide constructions for (variable-base) fixed-exponent and variable-exponent modular exponentiation puzzles and we show that the verifier only needs to perform a modest number of modular multiplications to construct and verify these puzzles. Given a modulus N, the costs incurred on the verifier in our puzzle are decreased by a factor of $\frac{|N|}{k}$ when compared to existing modular exponentiation puzzles, where k is a security parameter. For example, for a 1024-bit modulus N, $k = 80$, the verifier's cost is reduced by a factor of 12.

As a by-product, our puzzle can be used to efficiently verify the integrity of outsourced modular exponentiations (modulo a semi-prime). We further show how our puzzle can be integrated in protocols used for remote verification of computing performance and for DoS protection. We validate the performance of our puzzle through experiments on a large number of PlanetLab nodes [1].

The rest of the paper is organized as follows. In Section 2, we define client-puzzles and we introduce our assumptions based on RSA. In Section 3, we introduce our puzzles and we provide a security proof for their constructions. Section 4 outlines some applications that can benefit from our proposed scheme. In Section 5, we overview the related work and we conclude the paper in Section 6.

2 Preliminaries

2.1 Client Puzzle Properties

Here, we state the security notions of client puzzles (adapted from [14]).

Definition 1. *A client puzzle Puz is given by the following algorithms:*

- Setup *is a probabilistic polynomial time setup algorithm that is run by the puzzle generator. Given a security parameter k, it selects the key space $\mathcal{S}$, the hardness space $\mathcal{T}$, the string space $\mathcal{X}$, the puzzle instance space $\mathcal{I}$ and puzzle solution space $\mathcal{P}$. It then selects the puzzle parameters $params \leftarrow (\mathcal{S}, \mathcal{T}, \mathcal{X}, \mathcal{I}, \mathcal{P})$. The secret $s \in \mathcal{S}$ is kept private by the puzzle generator.*
- GenPuz *is a probabilistic polynomial time puzzle generation algorithm that is run by the puzzle generator. On input $s \in \mathcal{S}$, $Q \in \mathcal{T}$ and $a \in \mathcal{X}$, it outputs a puzzle instance $puz \in \mathcal{I}$.*
- FindSoln *is a probabilistic solution finding algorithm. On inputs $puz \in \mathcal{I}$ and a run time $\tau \in \mathbb{N}$, it outputs a potential solution $soln \in \mathcal{P}$ after at most τ clock cycles of execution.*
- VerAuth *is a puzzle authenticity verification algorithm. On inputs $s \in \mathcal{S}$ and $puz \in \mathcal{I}$, it outputs* true *or* false.
- VerSoln *is a deterministic solution verification algorithm. On inputs $s \in \mathcal{S}$, $puz \in \mathcal{I}$ and a solution $soln \in \mathcal{P}$, it outputs* true *or* false.

It is required that if $params \leftarrow \text{Setup}(k)$ *and* $puz \leftarrow \text{GenPuz}(s, Q, a)$ *where* $s \in \mathcal{S}$, $Q \in \mathcal{T}$ *and* $a \in \mathcal{X}$, *then* (1) $\text{VerAuth}(s, puz) = \text{true}$, (2) $\exists \tau \in \mathbb{N}$ *such that* $soln \leftarrow \text{FindSoln}(puz, \tau)$ *and* $\text{VerSoln}(s, puz, soln) = \text{true}$.

Definition 2. ***(Puzzle-unforgeability.)*** *A client puzzle Puz is* UF *(unforgeable) if the probability that any probabilistic polynomial-time adversary $\mathcal{M}$ succeeds in producing $\bar{Puz}$, such that $\bar{Puz}$ was not previously created by the puzzle generator and $VerAuth(\bar{Puz}) =$ true, is a negligible function of k.*

Definition 3. ***(Puzzle-difficulty.)*** *Let $\epsilon_{k,Q}(.)$ be a monotonically increasing function, where k is a security parameter and Q is a hardness parameter. A client-puzzle Puz is* $\text{DIFF}_{\text{k,Q}}$ *if for all $\tau \in \mathbb{N}$, for all security parameters $k \in \mathbb{N}$, for all $Q \in \mathbb{N}$, the success of any adversary $\mathcal{M}$, that is restricted to τ clock cycles of execution, is bounded by $\epsilon_{k,Q}(\tau)$ in solving Puz.*

2.2 Rivest's Repeated-Squaring Puzzle

In [43], Rivest *et al.* proposed a non-parallelizable time-lock puzzle based on repeated-squaring to enable time-release cryptography.

In this puzzle, the puzzle generator encrypts a message M into a ciphertext C as follows: $C = M + X^{a^t} \mod N$ given an integer X, an exponent a, a large integer t and an appropriate semi-prime modulus N. This computation can be performed efficiently using the trapdoor offered by Euler's function: $X^{a^t} \mod N \equiv X^{a^t \mod \phi(N)} \mod N$. On the other hand, to acquire M from C, the client needs to compute $X^{a^t} \mod N$ in $\log(a^t) \approx t$ modular multiplications.

When used as a client-puzzle (e.g., [44]), this puzzle is used such that the prover is required to compute $X^{a^t} \mod N$ given X, a, t and N. This computation is then verified by the puzzle generator through the trapdoor offered by Euler's function in $O(\log(N))$ modular multiplications.

2.3 RSA with a Small Private Exponent

The RSA cryptosystem [42] is the most widely used public-key cryptosystem. Let $N = pq$ be the product of two large and distinct primes and let e and d be inverses modulo $\phi(N) = (p-1)(q-1)$. Throughout the rest of the paper, we assume that p and q are balanced primes; that is, $|p| = |q|$. For $k \in \mathbb{N}^+$ ($\mathbb{N}^+ = \mathbb{N} - \{0\}$), the public RSA key e and the private RSA key d satisfy: $e \cdot d - 1 = k \cdot \phi(N)$.

It is known that RSA is insecure when $e \leq N^{1.875}$ and d is small [8, 10, 15–17, 20, 23, 25, 28, 33, 38, 50]. Existing attacks on this class of "weak" RSA keys are mostly based on Wiener's attack [50] and/or on Boneh and Durfee's attack [10]. Wiener's continued fraction attack can be used to efficiently factor N when $e \leq N$ and $d < N^{\frac{1}{4}-\epsilon}$ and Boneh and Durfee's lattice-based attack [10] shows that private exponents up to $N^{0.2929}$ are unsafe when $e < N^{1.875}$. Blömer *et al.* [8] further generalized Wiener's attack to factor N in polynomial time for every $e \leq N$ satisfying $ex + y \equiv 0 \mod \phi(N)$, where x and y are short. Gao[1] and Howgrave-Graham and Seifert [30] extended these attacks to factor N given several common modulus instances of RSA with $d < N^{0.4}$ and $e \leq N$.

2.4 Low-Cost Decryption in RSA

In this work, we consider RSA keys that do not belong to the weak class of RSA keys, yet enable low-cost decryption in RSA. More specifically, we consider the following class of RSA keys:

Class $\mathcal{A}$: *Class $\mathcal{A}$ is defined as the set of all RSA keys (N, e, d) where: $N = pq$, p and q are two large balanced primes, $e \geq N^2$ such that* $\gcd(e, \phi(N)) = 1$ *and d is small such that $ed - 1 \equiv 0 \mod \phi(N)$.*

When $(N, e, d) \in \mathcal{A}$, the fastest known algorithm that computes d from (N, e) runs exponentially in time with $|d|$. This hardness assumption on class $\mathcal{A}$ is based on the observations of Wiener [50] and Boneh *et al.* [10]. When $e \geq N^2$, all known attacks against small private RSA exponent are defeated. More specifically, the continued fraction algorithm [50], the lattice-based attack [10] and Coppersmith's attack [15,16] fail even when d is small (for the reason why, refer to Appendix A). For example, when $e \geq N^2$, $|d| \geq$ 80-bits, no known feasible algorithm can compute d from $(N, e) \in \mathcal{A}$, and therefore factor N. RSA keys that belong to $\mathcal{A}$ clearly do not optimize the cost of RSA encryption and signature schemes; when e is large, the cost of encryption and/or signature verification is prohibitively high, which explains why this class is not widely used in RSA.

Remark 1. Given the work of Blömer *et al.* [8], we can safely extend class $\mathcal{A}$ to the set of RSA keys that satisfy a generalized RSA key equation of the form $ex + y \equiv 0 \mod \phi(N)$, where $e \geq N^2$ and x, y are small (for the reason why, see Appendix B). Note that a special instance of this equation is the standard RSA equation, where $x = d$ and $y = -1$.

[1] Gao's unpublished attack is described by Howgrave-Graham and Seifert in [30].

Remark 2. One simple way to generate large public keys whose modular inverses are small is to pick d such that $|d|$ is small, and compute $e' = d^{-1} \mod \phi(N)$. Then, a large public key e is computed from e' as follows: $e = t\phi(N) + e'$, where $t \in \mathbb{N}^+$ and $t \approx N^2$. The verifier then deletes e' and publishes (N, e) [9].

Where appropriate, we also consider in this work the following class of RSA keys:

Class $\mathcal{B}$: *Class $\mathcal{B}$ is defined as the set of all RSA keys (N, e, d) where: $N = pq$, p and q are balanced large primes, $e \in \mathbb{N}^+$, $\gcd(e, \phi(N)) = 1$, $d_p \equiv d \mod p$ and $d_q \equiv d \mod q$ such that $d_p \neq d_q$, $d > N^{0.5}$ and $ed - 1 \equiv 0 \mod \phi(N)$.*

When $(N, e, d) \in \mathcal{B}$, the fastest known algorithm that computes d from (N, e) runs in $\min(\sqrt{d_p}, \sqrt{d_q})$. The use of RSA keys in class $\mathcal{B}$ is suggested by Wiener [50] and Boneh [10] to speed up RSA decryption[2]. Since decryptions are often generated modulo p and q separately and then combined using the Chinese Remainder Theorem (CRT) [11], Wiener proposes the use of a private key d such that both $d_q \equiv d \mod q$ and $d_p \equiv d \mod p$ are small ($d_p \neq d_q$). The best known attack against this scheme runs in $\min(\sqrt{d_p}, \sqrt{d_q})$ [10, 26].[3] When $|\min(\sqrt{d_q}, \sqrt{d_p})| \geq 80$ bits, $|N| = 1024$-bits, there exists no feasible algorithm that can compute d from $(N, e) \in \mathcal{B}$.

Remark 3. Throughout this paper, we consider RSA keys in the class $\mathcal{A} \cup \mathcal{B}$ as a building block to construct low-cost puzzles based on modular exponentiation. To simplify the description and analysis of our puzzles, we consider RSA keys in $\mathcal{A} \cup \mathcal{B}$ where the public exponent $e \geq N^2$. We point out, however, that our analysis also applies for all RSA keys in $\mathcal{A} \cup \mathcal{B}$.

3 Low-Cost Puzzles based on Modular Exponentiation

3.1 System and Attacker Model

We consider the following model. A verifier (puzzle generator) wants to verify that a prover performed a certain number of modular exponentiations (modulo a semi-prime) in a specified interval of time. For that purpose, the verifier requires that the prover runs a software on its machine (i.e., a modular exponentiation puzzle) for a specific amount of time. In some application scenarios, we will need to assume that the verifier and the prover can exchange authenticated messages over the communication channel. We assume, however, that the verifier does not have access to the prover's machine and thus cannot check the prover's environment; this includes the number of processors at the disposal of the prover, the connections established from the prover's machine, etc..

[2] This RSA variant is widely used in smart cards.

[3] Recently, Jochemsz *et al.* propose in [31] a polynomial attack on small private CRT-RSA exponents. This attack only works when $\min(d_p, d_q) \leq N^{0.073}$. However, in this case, brute-force search attacks would also be feasible on CRT-RSA.

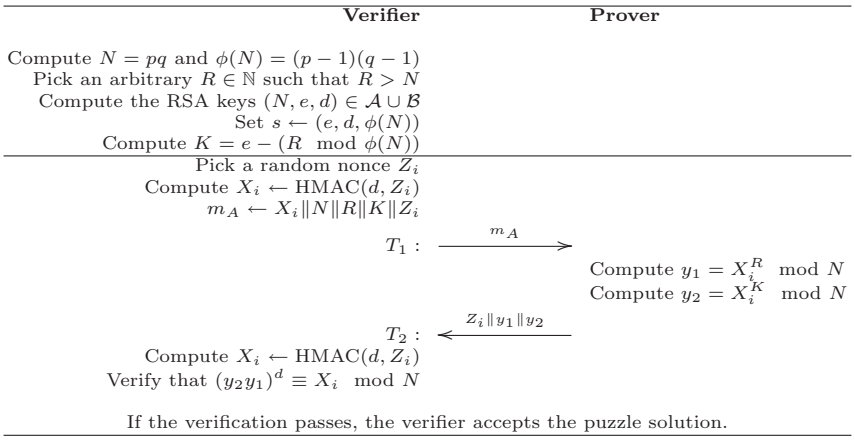

Fig. 1. Fixed-Exponent Puzzle based on Modular Exponentiation.

An untrusted prover constitutes the core of our attacker model. We assume that a prover possesses considerable technical skills by which it can efficiently analyze, decompile and/or modify executable code as necessary. More specifically, an untrusted prover has knowledge of the algorithm used for the computation and of the algorithm that is run by the verifier. We assume that untrusted provers are motivated to inflate their puzzle solving performance (i.e., untrusted provers have incentives to solve the puzzle in a faster time than what they can genuinely perform). However, we assume that provers are computationally bounded.

3.2 Low-Cost Fixed-Exponent Modular Exponentiation Puzzle

Here, we present our puzzle based on (variable-base) fixed-exponent modular exponentiation. In Section 3.3, we propose a variant puzzle based on variable-exponent modular exponentiation. Our puzzle is shown in Figure 1.

In the setup phase of our puzzle, the verifier picks two large balanced primes p and q (of sufficient size to prevent factoring of $N = pq$, e.g., $|p| = |q| \geq 512$-bits), computes $N = pq$ and $\phi(N) = (p-1)(q-1)$. Given N, the verifier also generates RSA keys (N, e, d) such that $(N, e, d) \in \mathcal{A}$, $|d| = k$, where k is a security parameter or $(N, e, d) \in \mathcal{B}$, where $|\min(\sqrt{d_p}, \sqrt{d_q})| = k$. The verifier also picks a puzzle difficulty $R \in \mathbb{N}$ and computes $K = e - (R \mod \phi(N))$. We show later that K will enable low-cost verification of the puzzle solution. (N, R, K) are public parameters that set the puzzle hardness and $s \leftarrow (e, d, \phi(N))$ is kept secret. Note that R needs to be larger than $\phi(N)$ to ensure the security of our scheme[4]. Typically, R is chosen such that $R \gg \phi(N)$ ($|R| \geq 100,000$ bits) to achieve a moderate runtime of the puzzle (in the order of tens of milliseconds, see Section 3.4). However, even in the case where the verifier would like to e.g., simply outsource the computation of an arbitrary $R' \leq \phi(N)$, this can be remedied by setting $R \leftarrow R' + t\phi(N)$, where $t \in \mathbb{N}^+$.

[4] This can be achieved by setting $R > N$.

- *Puzzle Generation:* In round i, the verifier generates $puz \leftarrow (X_i, Z_i, R, K, N)$,[5] where $X_i \leftarrow \text{HMAC}(d, Z_i)$. Here, Z_i is a nonce and $|X_i| \geq k$. In the sequel, we assume that $\text{HMAC}(X, Y)$ is a keyed collision-resistant pseudo-random function, where X is used as an input key.
- *Puzzle Solution:* Given puz, the prover computes $soln \leftarrow (y_1 = X_i^R \mod N, y_2 = X_i^K \mod N, Z_i)$.
- *Solution Verification:* Given $soln$, the verifier checks if $(y_2 y_1)^d \equiv X_i^{d(R+K)} \mod N \equiv X_i^{ed} \mod N \equiv X_i \mod N$.

Remark 4. Note that our puzzle is stateless; only a single value of the secret $s \leftarrow (e, d, \phi(N))$ is stored by the verifier regardless of the number of puzzles (instances) that the verifier generates. All the required data to solve a given puzzle is contained in puz, whereas the knowledge of s and $soln$ are sufficient to verify the puzzle solution $soln$. The uniqueness of each puzzle instance can be ensured by having GenPuz select Z_i a counter and increment Z_i in each puzzle instance.

Remark 5. When $R = 0$, the prover simply computes $y_2 = X_i^e \mod N$, and the verifier verifies the puzzle solution by computing y_2^d. Such a puzzle is then based on "standard" RSA. The major limitation of this "standard" RSA-based puzzle is that the choice of the puzzle difficulty (i.e., the exponent) is dependent on the choice of d and $\phi(N)$. This particularly hinders the construction of repeated-squaring puzzles (e.g., [43]) or the secure outsourcing of modular exponentiations for a given exponent.

Puzzle Construction and Verification Costs: In our puzzle, the verifier only needs to perform 1 HMAC operation (2 hashes) to construct the puzzle and a small number of modular multiplications (computing $(y_2 y_1)^d$) to verify the puzzle solution:

- $(N, e, d) \in \mathcal{A}$: In this case, the puzzle verification is performed in $O(\log d)$ modular multiplications. When $|d| = k$, the verifier's cost is reduced by a factor of $\frac{\log N}{\log d} = \frac{|N|}{k}$, when compared to the original repeated-squaring puzzle [43]. When $|N| = 1024, k = 80$, the puzzle verification cost could be as low as $\frac{3}{2}80 = 120$ modular multiplications[6] and the average improvement gain in the puzzle solution verification is almost 12 (i.e., $\frac{1.5\times 1024}{1.5\times 80}$). Similarly, when $|N| = 2048$, $k = 112$, the average improvement gain increases to 18.
- $(N, e, d) \in \mathcal{B}$: In this case, the puzzle verification is performed in $O(\log(d_p) + \log(d_q))$ modular multiplications using the CRT. When $|\min(\sqrt{d_p}, \sqrt{d_q})| = k$, the verifier's cost is reduced by a factor of $\frac{\log N}{2 \log d} = \frac{|N|}{4k}$, when compared to the original repeated-squaring puzzle [43].

[5] When R is very large, the verifier can reduce the communication costs by sending $r \ll R$, such that $R = F(r)$, where $F(r)$ is an expansion function of r.

[6] On average, the computation of $X^d \mod N$ requires $1.5 \log d$ modular multiplications [35].

Prime number generation (i.e., computing N) and the pre-computation of e and d are generally expensive operations for the verifier; however, this computation is performed only once at the setup phase[7] and (N, e, d) are subsequently used for all the puzzles generated by the verifier.

Security Analysis: To analyze the security of our scheme, we first show that it is computationally infeasible for an adversary to acquire the secret s held by the verifier in our puzzle. Based on this, we show that an adversary needs to perform at least $O(\log R)$ modular multiplications to compute the solution *soln* to a puzzle instance *puz* such that $\text{VerSoln}(s, puz, soln) = \text{true}$.

We use the following game $\text{Exec}_{\mathcal{M}}(k)$ between a challenger and a probabilistic polynomial time (p.p.t.) adversary $\mathcal{M}$:

- The challenger runs Setup on input k to obtain $N = pq$ chosen uniformly at random from $\mathcal{N}$, d chosen uniformly at random from $\{2^k..2^{k+1}\}$ and computes e such that $(N, e, d) \in \mathcal{A} \cup \mathcal{B}$. The challenger, then stores the secret $s \leftarrow (e, d, \phi(N))$. The challenger further picks $R > N$ chosen uniformly at random from $\mathcal{R}$ and computes K as shown in Figure 1.
- The adversary $\mathcal{M}$ gets to make as many $\text{CreatePuz}(Z_i)$ queries as it likes. In response, the challenger *(1)* creates $puz \leftarrow (X_i, Z_i, R, K, N)$ as shown in Figure 1, *(2)* computes *soln* such that $\text{VerSoln}(s, puz, soln) = \text{true}$ and *(3)* outputs $(puz, soln)$.

Adversary $\mathcal{M}$ terminates the game by outputting an integer C. We say that $\mathcal{M}$ wins $\text{Exec}_{\mathcal{M}}(k)$ if $C \equiv 0 \mod \phi(N)$ (i.e., if $\mathcal{M}$ computes a multiple of $\phi(N)$). In this case, we set the output of $\text{Exec}_{\mathcal{M}}(k)$ to be 1 and otherwise to 0. We then define the success of $\mathcal{M}$ as $\text{Succ}_{\mathcal{M}}(k) = Pr[\text{Exec}_{\mathcal{M}}(k) = 1]$.

Theorem 1. *Computing a multiple of $\phi(N)$ and, in particular, computing d given (N, e) is computationally as hard as factoring (see [40] for the proof).*

Lemma 1. *$(N, R + K, d) \in \mathcal{A} \cup \mathcal{B}$ if $(N, e, d) \in \mathcal{A} \cup \mathcal{B}$.*

Proof. Let $(N, e, d) \in \mathcal{A} \cup \mathcal{B}$ satisfy the RSA key equation: $ed - 1 \equiv 0 \mod \phi(N)$. Recall that e is kept secret by the challenger. Since $K = e - (R \mod \phi(N))$, then $\exists t_1 \in \mathbb{N}^+$ (since $R > N$) such that $R + K = e + t_1\phi(N)$. This means that $d(R + K) \equiv de \equiv 1 \mod \phi(N)$.

Given Theorem 1, computing e from $(R + K)$ is computationally as hard as factoring[8]. Since d is the modular inverse of e, d is equally the modular inverse of $(R + K)$. More specifically, it is easy to see that since $(N, e, d) \in \mathcal{A} \cup \mathcal{B}$, then $(N, (R + K), d)$ are RSA keys in $\mathcal{A} \cup \mathcal{B}$ (since $(R + K) > e$).

Lemma 2. *For any p.p.t. adversary $\mathcal{M}$, $\text{Succ}_{\mathcal{M}}(k)$ is a negligible function of k.*

[7] Note that the computational load incurred by prime number generation equally applies to all protocols that make use of modular exponentiation or repeated-squaring (e.g., [43, 49]).

[8] $(R + K - e)$ is a multiple of $\phi(N)$.

Proof. We show that if $\mathcal{M}$ can compute a multiple of $\phi(N)$ in the $\text{Exec}_{\mathcal{M}}(k)$ game, then we can construct a polynomial-time algorithm that uses $\mathcal{M}$ as a subroutine to solve the RSA problem in $\mathcal{A} \cup \mathcal{B}$, i.e., to compute $\bar{d}$ given a public RSA key $(\bar{N}, \bar{e})$ where $(\bar{N}, \bar{e}, \bar{d}) \in \mathcal{A} \cup \mathcal{B}$.

Let $\mathcal{M}$ be a p.p.t. adversary that outputs a multiple of $\phi(N)$ in the game $\text{Exec}_{\mathcal{M}}(k)$ with probability $\text{Succ}_{\mathcal{M}}(k)$. In the $\text{Exec}_{\mathcal{M}}(k)$ game, let $R + K = e'$. Recall that in $\text{Exec}_{\mathcal{M}}(k)$, $K = e - R \mod \phi(N)$, where e is chosen uniformly at random from $\mathcal{A} \cup \mathcal{B}$ and $R > N$. Given this, note that $e' > e + \phi(N)$ and $K \geq N^2 - \phi(N)$; this suggests that $R + K = e' > R + N^2 - \phi(N)$ and therefore $R < e' - N^2$.

Let $(\bar{N}, \bar{e}, \bar{d}) \in \mathcal{A} \cup \mathcal{B}$, where $N \in \mathcal{N}$, $d \in [2^k, .., 2^{k+1}[$, $\bar{e} \geq N^2 + \phi(N)$. Then, we construct a polynomial-time algorithm $\mathcal{E}$ that interacts with $\mathcal{M}$ as follows:

- Given the public key $(\bar{N}, \bar{e})$, $\mathcal{E}$ picks $\bar{R}$ at random from $\{N + 1, .., \bar{e} - N^2\}$.
- $\mathcal{E}$ computes $\bar{K} = \bar{e} - \bar{R}$ and constructs a transcript $\bar{T}$ that is composed of a number of tuples of the form $(\bar{X}_i, \bar{Z}_i, \bar{R}, \bar{K}, \bar{N}, \bar{X}_i{}^{\bar{R}} \mod \bar{N}, \bar{X}_i{}^{\bar{K}} \mod \bar{N})$, $i \in \mathbb{N}$, where $\bar{X}_i$ is a pseudorandom string that has a similar distribution as HMAC(.) and $\bar{Z}_i$ is a counter.

Note that since $\bar{R} > \bar{N}$, $\exists t_1 \in \mathbb{N}^+$ such that $\bar{R} - t_1\phi(\bar{N}) = (\bar{R} \mod \phi(\bar{N}))$. Let $\bar{e_1} = \bar{e} - t_1\phi(\bar{N})$. It is easy to see in this case that $\bar{K} = \bar{e_1} - (\bar{R} \mod \phi(\bar{N}))$. Furthermore, since $\bar{e_1} \equiv \bar{e} \mod \phi(\bar{N})$, then d is a modular inverse of $\bar{e_1}$. We point out that since $\bar{e_1} = \bar{e} - t_1\phi(\bar{N}) = \bar{e} - \bar{R} + (\bar{R} \mod \phi(\bar{N}))$, then $\bar{e_1} > \bar{e} - \bar{R} \geq N^2$, since by construction $\bar{R} \leq \bar{e} - N^2$. Therefore, $(\bar{N}, \bar{e_1}, \bar{d}) \in \mathcal{A} \cup \mathcal{B}$.

Given this, it is easy to see that the view of $\mathcal{M}$ when run as a subroutine by $\mathcal{E}$ is distributed identically to the view of $\mathcal{M}$ in the game $\text{Exec}_{\mathcal{M}}(k)$. Recall that in $\text{Exec}_{\mathcal{M}}(k)$, X_i is a pseudorandom string, $K = e - R \mod \phi(N)$, where e is a secret such that $(N, e, d) \in \mathcal{A} \cup \mathcal{B}$ and $(N, R + K, d) \in \mathcal{A} \cup \mathcal{B}$.

Therefore, if $\mathcal{M}$ can compute a multiple of $\phi(N)$ in the $\text{Exec}_{\mathcal{M}}(k)$ game, then it can solve the above RSA problem. By the hardness assumption on $\mathcal{A}$ and $\mathcal{B}$, it is computationally infeasible for $\mathcal{M}$ to compute $\bar{d}$, or equivalently a multiple of $\phi(\bar{N})$ (Theorem 1), from $(\bar{N}, \bar{e})$ when $(\bar{N}, \bar{e}, \bar{d}) \in \mathcal{A} \cup \mathcal{B}$. Therefore, $\text{Succ}_{\mathcal{M}}(k)$ is negligible, thus concluding the proof.

Given this, we can show that our puzzle construction is both unforgeable (UF) and difficult ($\text{DIFF}_{\text{k,R}}$).

Corollary 1. *The puzzle construction of Figure 1 is* UF.

Proof Sketch: Given a puzzle instance $puz \leftarrow (X_i, Z_i, R, K, N)$, VerAuth($s$, puz) = true if and only if $X_i \leftarrow \text{HMAC}(d, Z_i)$.

Therefore, the only viable way for $\mathcal{M}$ to construct $p\bar{u}z \leftarrow (\bar{X}_i, \bar{Z}_i, R, K, N)$ such that VerAuth($p\bar{u}z$) = true and $p\bar{u}z$, $\bar{X}_i$, $\bar{Z}_i$ were not previously created by the challenger is to construct $(\bar{X}_i, \bar{Z}_i)$ such that $\bar{X}_i \leftarrow \text{HMAC}(d, \bar{Z}_i)$. Since HMAC(.) is a pseudorandom collision-resistant function, $\mathcal{M}$ cannot construct $(\bar{X}_i, \bar{Z}_i)$ without the knowledge of d. Following from Lemma 2, the success probability for $\mathcal{M}$ in acquiring d from our puzzle – and therefore constructing $p\bar{u}z$ such that VerAuth($p\bar{u}z$) = true – is bounded by $O(2^{-k})$.

Corollary 2. *The puzzle construction of Figure 1 is* $\mathrm{DIFF}_{\mathrm{k,R}}$.

Proof Sketch: Following from Lemma 2, it is computationally infeasible for $\mathcal{M}$ to compute a multiple of $\phi(N)$ given our puzzle. Furthermore, $\mathcal{M}$ cannot pre-compute the solution of the puzzle since it cannot predict X_i ($|X_i| \geq k$) nor the outcome of $y_2 y_1$ (since e is kept secret by the verifier).

The fastest known way for $\mathcal{M}$ to solve our puzzle is to compute y_1 and y_2 correctly. Modular Multiplication is an inherently sequential process [43]. The running time of the fastest known algorithm for modular exponentiation is linear in the size of the exponent. Although $\mathcal{M}$ might try to parallelize the computation of y_1 and/or y_2, the parallelization advantage is expected to be negligible[9] [43,44].

Note that $\mathcal{M}$ might try to perform the computation of y_1 and y_2, in parallel, using different machines at its disposal. In typical cases, $R \gg K$; this means that the computation of y_1 and y_2 requires at least $O(\log R)$ sequential modular multiplications. We point out that the verifier can prevent the separate computation of y_1 and y_2, by sending K to the prover once it receives y_1 (see Figure 2).

$\mathcal{M}$ can equally try to compute y_1 and/or y_2 through intermediate results that it previously computed (or intercepted) (e.g., when the base X_i is the result of a multiplication of two previously used numbers). This also applies to the original time-lock puzzle proposed in [43]; this can be remedied, with high probability, by setting $|\mathrm{HMAC}(.)| \gg |Z_i|$.

Given this, the success of $\mathcal{M}$ – restricted to τ clock cycles of execution – in solving our puzzle is bounded by $\epsilon_{k,R}(\tau) = \min(\lfloor \frac{\tau}{\log R} \rfloor + O(2^{-k}), 1)$; $\mathcal{M}$ needs to perform at least $\tau = \log(R)$ clock cycles of execution to solve our puzzle.

3.3 Low-Cost Variable-Exponent Modular Exponentiation Puzzle

In some settings, the verifier might need to change the puzzle difficulty (i.e., the exponent) "on the fly" (e.g., when subject to DoS attacks). We briefly discuss how this can be achieved based on the proposed fixed-exponent puzzle.

Our variable-exponent puzzle and the related protocol are depicted in Figure 2. Similar to the fixed-exponent puzzle (Figure 1), in round i, the verifier creates the RSA keys $(N, e, d) \in \mathcal{A} \cup \mathcal{B}$, picks $Z_i \in \mathbb{N}$ and computes $X_i \leftarrow \mathrm{HMAC}(d, Z_i)$. Here, in addition, the verifier computes $v_i \leftarrow \mathrm{HMAC}(d, X_i)$ such that $|v_i| = k$ and $\gcd(v_i, d) = 1$.[10]

[9] $\mathcal{M}$ might try to parallelize the multiplication of large numbers by splitting the multiplicands into smaller "words" and involving other processors in the multiplication of these words. Further details about this process can be found in [36]. However, this attack incurs a significant communication overhead that prevents an $\mathcal{M}$ from gaining any substantial speedup; given a large number of squaring rounds, the RTT between the cooperating processors needs to be in the order of few nanoseconds to achieve even a modest speedup.

[10] The probability that any number is coprime with d is $\frac{6}{\pi^2} \approx 0.61$. Therefore, only two choices are sufficient, on average, to create such a v_i (i.e., if $\gcd(\mathrm{HMAC}(d, X_i)), d) \neq 1$, then with high probability $\gcd(\mathrm{HMAC}(d, X_{i+1})), d) = 1$).

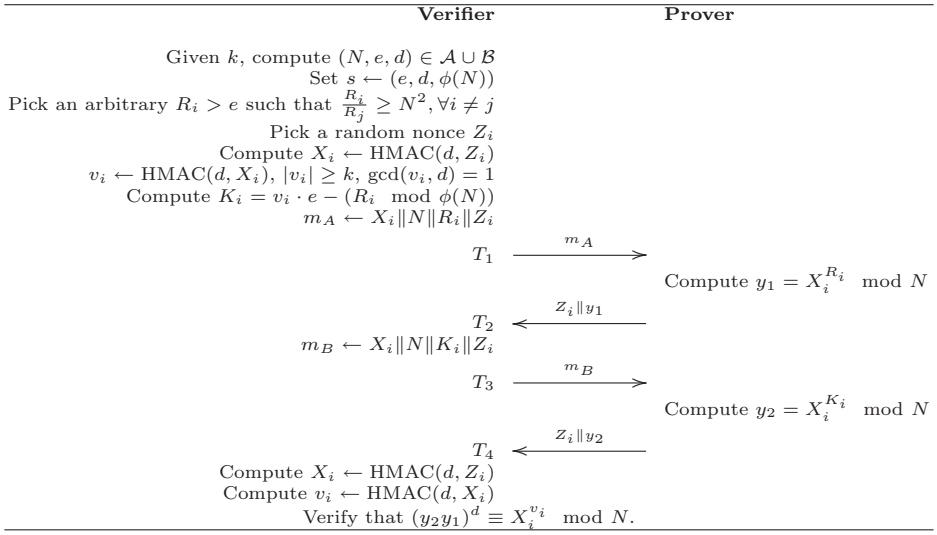

Fig. 2. Variable-Exponent Puzzle based on Modular Exponentiation. Note that y_1 and y_2 could be also transmitted in the same message. The separate transmission of y_1 and y_2, however, prevents the computation of y_1 and y_2 in parallel and enables the use of this puzzle to remotely verify the computing performance of devices (see Section 4.2).

The puzzle instance at round i is then comprised of the tuple $puz \leftarrow (X_i, Z_i, N, R_i, K_i)$, where $K_i = v_i e - (R_i \mod \phi(N))$, and $R_i \in \mathbb{N}$. Its solution is $soln \leftarrow (Z_i, X_i^{R_i} \mod N, X_i^{K_i} \mod N)$. To verify $soln$, the verifier checks if $(y_2 y_1)^d \equiv X_i^{v_i} \mod N$.

It is easy to see that the cost incurred on the verifier in this puzzle exceeds that of the fixed-exponent puzzle by $|v_i| = k = 80$ modular multiplications (mainly in puzzle solution verification). For instance, when $(N, e, d) \in \mathcal{A}$, $soln$ can be verified in 240 modular multiplication; the verification gain when compared to existing modular exponentiation puzzles is then $\frac{1536}{240} \approx 7$, given a 1024-bit N.

Corollary 3. *The puzzle construction of Figure 2 is* UF *and* $\text{DIFF}_{\text{k,R_i}}$ *when* (1) $(N, e, d) \in \mathcal{A}$ *and* $R_i > e$ *such that* $\frac{R_i}{R_j} \geq N^2, \forall i \neq j$, *or* (2) $(N, e, d) \in \mathcal{B}$.

Proof Sketch: Due to lack of space, we only provide the main intuition behind the proof.

Consider a variant of the aforementioned $\text{Exec}_{\mathcal{M}}(k)$ game where the transcript of interaction T between the adversary $\mathcal{M}$ and the challenger is composed of a number of tuples $(X_i, Z_i, R_i, K_i, N, X_i^{R_i} \mod N, X_i^{K_i} \mod N)$, $i \in \mathbb{N}$.

Similar to the analysis in Lemma 2, we can show that if $\mathcal{M}$ can compute a multiple of $\phi(N)$ in this variant $\text{Exec}_{\mathcal{M}}(k)$ game, then it can compute a multiple of $\phi(N)$ given several instances of the generic RSA key equation $e_i x_i + y_i \equiv 0 \mod \phi(N)$ with common modulus and unknown x_i, y_i, where $e_i = R_i + K_i$, $x_i = d$ and $y_i = -v_i$. Note that $v_i \neq v_j, \forall i \neq j$. This is especially important for

	Verifier Cost	**Prover Cost**
Repeated-Squaring [43]	1 modulus, 1 mul. $O(\log(N))$ mod. mul.	$O(\log R)$ mod. mul.
Fixed Exponent **$\mathcal{A}$-Puzzle**	1 modulus, 1 HMAC $O(\log(d))$ mod. mul. $^{(*)}$	$O(\log R) + O(\log(N))$ mod. mul.
Variable Exponent **$\mathcal{A}$-Puzzle**	1 modulus, 2 HMAC $O(\log(d) + \log(v))$ mod. mul. $^{(*)}$	$O(\log R) + O(\log(N))$ mod. mul.
Fixed Exponent **$\mathcal{B}$-Puzzle**	1 modulus, 1 HMAC $O(\log d^2)$ mod. mul. $^{(*)}$	$O(\log R) + O(\log(N))$ mod. mul.
Variable Exponent **$\mathcal{B}$-Puzzle**	1 modulus, 2 HMAC $O(\log d^2 + \log(v))$ mod. mul. $^{(*)}$	$O(\log R) + O(\log(N))$ mod. mul.

Table 1. Construction and Verification Costs of Puzzles. "Mod. Mul." denotes modular multiplication and "Mul." refers to multiplication. $\mathcal{B}$-Puzzle and $\mathcal{A}$-Puzzle refer to our proposed puzzle created using classes $\mathcal{B}$ and $\mathcal{A}$, respectively, of RSA keys. $^{(*)}$ Note that $d \ll N$, $v \ll N$; $|v| = |d| = k \geq 80$.

the security of our puzzle. Otherwise, $\mathcal{M}$ can compute a multiple of $\phi(N)$ solely from R_i and R_j ($(R_i - R_j) \equiv 0 \mod \phi(N)$).

When $(N, e, d) \in \mathcal{A}$ and $R_i > e$ such that $\frac{R_i}{R_j} \geq N^2, \forall i \neq j$, then $\frac{e_i}{e_j} = \frac{R_i + K_i}{(R_j + K_j} > N, \forall i \neq j$. In this case, all existing attacks on common modulus instances of RSA are defeated (refer to Remark 1 and the related Appendix B); the best known algorithm that computes $\phi(N)$ from (N, e_i) runs exponentially in time in $|x_i y_i| = |dv_i|$ since d and v_i are in lowest terms by construction (i.e., $\gcd(d, v_i) = 1$). In our case, $|dv_i| \geq 2k = 160$. We conclude that it is computationally infeasible for $\mathcal{M}$ to compute a multiple of $\phi(N)$ from T. Similarly, when $(N, e, d) \in \mathcal{B}$, $d = |x_i| > N^{0.5}$ [10,50], there exists no polynomial-time algorithm that can factor N in this case [27].

Similar to Corollaries 1 and 2, it can be shown that the puzzle construction of Figure 2 is UF and $\text{DIFF}_{\text{k,R}_\text{i}}$.

3.4 Performance Evaluation

Table 1 summarizes the costs incurred in our puzzles when compared with those in the repeated-squaring puzzle of [43]. To the best of our knowledge, there are no other proposed non-parallelizable puzzles that are based on modular exponentiation.

To evaluate the performance of our puzzle, we implemented it in JAVA on four different workstations. We evaluate the performance of our scheme on various other processors in Section 4.2. In our implementation, we used built-in JAVA functions for prime number generation, repeated-squaring using addition chains, etc.. While a faster implementation of our scheme could be achieved using lower-level programming and/or specialized hardware or software, we aim to demonstrate in this work the feasibility of our proposal using available standard algorithms and programming tools. Our findings (Table 2) show that our schemes considerably reduce the cost incurred on the generator of modular exponentiation puzzles.

# Squaring (Size of R in bits)	Puzzle Runtime	Verification Time $\mathcal{A}$-Puzzle	Verification Time $\mathcal{B}$-Puzzle	Verification Time of [43]
6500000	154.067 s	1.89 ms	7.56 ms	24.3 ms
6500000	172.174 s	2.11 ms	8.5 ms	27.12 ms
6500000	170.611 s	2.1 ms	8.4 ms	26.9 ms
6500000	165.034 s	2.03 ms	8.12 ms	26 ms

Table 2. Implementation results on four different workstations equipped with Intel(R) Core(TM)2 Duo CPU T7500 processor running at 2.20 GHz. Here, N is 1024-bit composite integer, $k = 80$. We conducted our measurements over the LAN (max RTT = 100 ms). Our results are averaged over 10 distinct measurements. The puzzle verification time is interpolated from the number of squarings per second on each machine.

4 Applications

4.1 Efficient Resilience to DoS Attacks

A natural application of our puzzles lies in the area of protection against DoS attacks. In this context, an online server requires that its clients solve our puzzles before attending to their requests in order to prevent DoS attacks.

When used in DoS protection, it is important, however, that the server ensures that puzzle instances and solution pairs are used only once. To achieve this, the server should filter out resubmitted correctly solved puzzles and solution pairs [14, 32]. In our case, the storage is minimized since the server can simply store the hash of the nonce Z_i that corresponds to the most recent solved puzzle (where Z_i is a counter). The verifier will then accept to verify only recent puzzles.

4.2 Remote Verification of Computing Performance

To cope with the advances in processing power, the computing community is relying on the use of benchmarks. While several benchmarks [2, 13, 19, 44] were proposed as a mean to evaluate a processor's computing power, most of these benchmarks are parallelizable (see Section 5).

Based on our variable-exponent puzzle (Figure 2), we construct a secure benchmark that enables any machine (even with modest computation power, e.g., a PDA device) to remotely *upper-bound* the computing performance of single-processors. Our benchmark differs from the puzzle in Figure 2 as follows: upon reception of y_1, the verifier estimates the number of squarings per second: $S = \frac{R_i}{T_2 - T_1}$ of the prover's machine. This estimate is accepted by the verifier if, after receiving y_2 at time T_4: *(1)* $(T_4 - T_3) \leq \epsilon \cdot (T_2 - T_1)$, given a negligible ϵ and *(2)* $(y_2 y_1)^d \equiv X^{v_i} \mod N$.

Corollary 4. *Given the puzzle depicted in Figure 2, the success of a p.p.t. adversary $\mathcal{M}$ in inflating the number of squarings that it can perform per second by more than a small ϵ is negligible.*

Proof Sketch: Recall that our puzzle is UF and $\mathrm{DIFF}_{\mathrm{k,R_i}}$. Therefore, the only viable method for $\mathcal{M}$ to inflate its performance claim is to send $\bar{y_1}$, chosen at random, ahead of time, compute y_1 correctly and distribute the computation of

the corresponding $\bar{y_2}$ and y_2, such that $\bar{y_2}\bar{y_1} \equiv y_2 y_1 \mod N$, to other nodes at its disposal. This would enable $\mathcal{M}$ to decrease the measured time corresponding to the computation of $O(\log R_i)$ modular multiplications by $\Delta = (T_4 - T_3)$ time units (Δ includes the communication delay D of the path between the verifier and $\mathcal{M}$). However, this is countered by the fact that the verifier does not accept the prover's performance claim unless $(T_4 - T_3) \leq \epsilon \cdot (T_2 - T_1)$.

In this case, the maximum performance claim that $\mathcal{M}$ can make is $S_{max} = \frac{|R_i|}{(1-\epsilon)\cdot(T_2-T_1)}$. Note that ϵ is interpolated from the measured number of squarings per second S; if it takes $(T_2 - T_1)$ time units for $\mathcal{M}$ to perform $\log(R_i)$ modular multiplications, then the computation of y_2 can be upper-bounded by choosing $\epsilon = \frac{\log(e-N)}{\log(R_i)}$. For a 1024-bit modulus N ($|\phi(N)| \approx |N|$), $|e| \approx |N^2|$ and $|R_i| > 100,000$, then $\epsilon \simeq \frac{S_{max}}{S} \simeq 0.03$ squarings per second.

Our protocol finds applicability in a multitude of application domains. For example, our benchmark can be used in online distributed computing applications (e.g., [3]) or in the secure ranking of supercomputers (e.g., [4]) to prevent possible frauds in performance claims[11].

We evaluated our benchmark on various processors running on 12 different PlanetLab nodes [1] (refer to Section 3.4 for implementation details). Our findings (see Table 3) suggest that our proposed benchmark reflects well the performance of various processors.

CPU Description	Idle CPU	S
Intel(R) Pentium(R) D 3.20GHz	6.40%	7.48
Intel(R) Pentium(R) D 3.00GHz	26.20%	15.24
Intel(R) Pentium(R) 4 3.20GHz	30.70%	15.81
Intel(R) Pentium(R) D 3.40GHz	14.10%	18.22
Intel(R) Xeon(R) 3060 2.40GHz	46.60%	28.01
Intel(R) Pentium(R) D 3.20GHz	20.00%	29.35
Intel(R) Xeon(R) 3075 2.66GHz	19.70%	29.72
Intel(R) Pentium(R) 4 3.06GHz	92.00%	31.72
Intel(R) Duo E6550 2.33GHz	63.80%	36.05
Intel(R) Duo T7500 2.20GHz	76.00%	38.11
Intel(R) Xeon(R) X3220 2.40GHz	73.30%	41.67
Intel(R) Xeon(R) E5420 2.50GHz	87.70%	50.97

Table 3. Implementation results on 12 different PlanetLab Nodes. S refers to the number of squarings per ms.

5 Related Work

Client puzzles found their application in several domains (e.g., prevention against DoS attacks [21, 48], protection from connection depletion attacks [32], protection against collusion [41]). Several computational puzzles have been proposed in the recent years [43, 45, 49]. However, most of these puzzles are parallelizable; a comprehensive survey of existing client puzzles can be found in [45]. In [43], Rivest *et al.* proposed a non-parallelizable puzzle based on repeated-squaring to enable time-release cryptography. The drawback of this scheme, if used for DoS protection, is that it incurs an expensive cost on the puzzle generator. Wang *et al.* propose in [47] a scheme that enables the server to adjust the puzzle difficulty in

[11] For instance, a supercomputer, connected to a hidden processor cluster, can inflate its performance claims by involving these other processors in the construction of the benchmark's solution. The literature contains a significant number of similar "anecdotes" where both individuals and manufacturers have tendencies to exaggerate their computing performance (e.g., [5, 6]).

the presence of an adversary whose computing power is unknown. In [14], Chen *et al.* provide a formal model for the security of client-puzzles. In this work, we use their model as a building block for analyzing the security of our proposed puzzle. Several other contributions address the problem of secure outsourcing of computations to untrusted servers (e.g., [7, 12]). Clarke *et al.* present protocols for speeding up exponentiation using untrusted servers in [46]. In [29], Hohenberger *et al.* describe a scheme to outsource cryptographic computations where the verifier can use two untrusted exponentiation programs to assist him in the computations. Memory-bound puzzles were proposed in [22, 37] to overcome the limitations of existing computational puzzles. However, memory-bound puzzles cannot entirely substitute their computational counterpart e.g., in applications where the client's memory is limited (e.g., PDA devices) or to evaluate the computing performance of devices, etc.. Other protocols for creating secure benchmarks to evaluate a machine's computing performance were also proposed [44]; these benchmarks can however be easily parallelized [18, 34, 39].

6 Conclusion

In this paper, we proposed low-cost fixed-exponent and variable-exponent puzzles based on modular exponentiation. Given a modulus N, the costs incurred on the verifier in our puzzle are decreased by a factor of $\frac{|N|}{k}$ when compared to existing modular exponentiation puzzles, where k is a security parameter. Our constructions are based on a reasonable intractability assumption: essentially the difficulty of computing a small private exponent in RSA (or CRT-RSA) when the public key is larger by several orders of magnitude than the semi-prime modulus. As a by-product, our puzzle can be used to efficiently verify the integrity of outsourced exponentiations modular a semi-prime. We further showed how our puzzle can be integrated in a number of protocols, including those used for the remote verification of computing performance of devices and for protection from DoS attacks.

7 Acknowledgments

The authors thank Rolf Wagner for implementing the repeated-squaring protocols. The authors also thank Cas Cremers, Patrick Schaller, Stephano Tessaro and Divesh Aggarwal for helpful discussions. Finally, the authors would like to thank the anonymous reviewers for their insightful suggestions and feedback.

References

1. PlanetLab, An open platform for developing, deploying, and accessing planetary-scale services, `http://www.planet-lab.org/`.
2. Linpack, `http://www.netlib.org/linpack/`.
3. Distributed.Net, `http://distributed.net/`.

4. TOP500 Supercomputing Sites, http://www.top500.org/.
5. Conroe Performance Claim being Busted, http://sharikou.blogspot.com/2006/04/conroe-performance-claim-being-busted.html.
6. Computer Software Manufacturer agrees to settle Charges, http://www.ftc.gov/opa/1996/07/softram.shtm.
7. M. J. Atallah, K. N. Pantazopoulos, John R. Rice, and Eugene H. Spafford. Secure Outsourcing of Scientific Computations. In *Advances in Computers*, 2001.
8. J. Blomer and A. May. Low Secret Exponent RSA Revisited. In *Cryptography and Lattice Conference (CaLC 2001)*, 2001.
9. D. Boneh. Twenty Years of Attacks on the RSA Cryptosystem. In *Notices of the American Mathematical Society (AMS)*, 1999.
10. D. Boneh and G. Durfee. Cryptanalysis of RSA with private key d less than $N^{0.292}$. In *IEEE Transactions on Information Theory*, pages 1339–1349, 2000.
11. D. Boneh and H. Schackam. Fast Variants of RSA. In *CryptoBytes*, 2002.
12. J. Burns and C.J. Mitchell. On parameter selection for server-aided RSA computation schemes. In *IEEE Transactions on Computers*, 1994.
13. J. Cai, A. Nerurkar, and M. Wu. The Design of Uncheatable Benchmarks Using Complexity Theory. Available from ftp://ftp.cs.buffalo.edu/pub/tech-reports/./97-10.ps.Z.
14. L. Chen, P. Morrissey, N. Smart, and B. Warinschi. Security Notions and Generic Constructions for Client Puzzles. In *Advances in Cryptology (Asiacrypt'09)*, 2009.
15. D. Coppersmith. Finding a Small Root of a Univariate Modular Equation. In *Advances in Cryptology, Eurocrypt*, pages 155–165, 1996.
16. D. Coppersmith. Small solutions to polynomial equations and low exponent vulnerabilities. In *Journal of Cryptology*, pages 223–260, 1997.
17. D. Coppersmith, M. Franklin, J. Patarin, and M. Reiter. Low-exponent RSA with related messages. In *Advances in Cryptology (EUROCRYPT)*, 1996.
18. Z. Cui-xiang, H. Guo-qiang, and H. Ming-he. Some New Parallel Fast Fourier Transform Algorithms. In *Proceedings of Parallel and Distributed Computing, Applications and Technologies*, 2005.
19. H.J. Curnow and B.A. Wichman. A Synthetic Benchmark. In *Computer Journal*, 1976.
20. B. de Weger. Cryptanalysis of RSA with small prime difference. In *Applicable Algebra in Engineering, Communication and Computing*, 2002.
21. D. Dean and A. Stubblefield. Using client puzzles to protect TLS. In *Proceedings of the USENIX Security Symposium*, 2001.
22. S. Doshi, F. Monrose, and A. Rubin. Efficient Memory Bound Puzzles using Pattern Databases. In *Proceedings of the International Conference on Applied Cryptography and Network Security (ACNS)*, 2006.
23. G. Durfee and P. Nguyen. Cryptanalysis of the RSA Schemes with Short Secret Exponent from Asiacrypt 99. In *Advances in Cryptology, Asiacrypt*, 2000.
24. Y. Gao. Efficient Trapdoor-Based Client Puzzle System against DoS Attacks. 2005.
25. J. Hastad. Solving Simultaneous Modular Equations of Low Degree. In *Siam J. Computing*, 1988.
26. M. J. Hinek. Cryptanalysis of RSA and its variants. In *Chapman & Hall/CRC, cryptography and network security*, 2009.
27. M. J. Hinek and C. C. Y. Lam. Common Modulus Attacks on Small Private Exponent RSA and Some Fast Variants (in Practice). In *Cryptology ePrint Archive*, 2009.
28. M.J. Hinek. Another Look at Small RSA Exponents. In *Topics in Cryptology*, 2006.

29. S. Hohenberger and A. Lysyanskaya. How To Securely Outsource Cryptographic Computations. In *Theory of Cryptography Conference, LNCS, Springer*, volume 3378, pages 264–282, 2005.
30. N. Howgrave-Graham and J. P. Seifert. Extending Wiener's Attack in the Presence of Many Decrypting Exponents. In *Proceedings of the International Exhibition and Congress on Secure Networking*, 1999.
31. E. Jochemsz and A. May. A Polynomial Time Attack on RSA with Private CRT-Exponents Smaller Than $N^{0.073}$. In *Advances in Cryptology (Crypto)*, 2007.
32. A. Juels and J. Brainard. Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks. In *Proceedings of NDSS*, 1999.
33. S. Katzenbeisser. Recent Advanves in RSA Cryptography. In *Advances in Information Security (Volume 3)*, 2001.
34. L. Keqin. Scalable Parallel Matrix Multiplication on Distributed Memory-Parallel Computers. In *Proceedings of IPDPS*, 2000.
35. N. Koblitz. A Course in Number Theory. 1987.
36. C. Kaya Koc, T. Acar, and B.S. Kaliski. Analyzing and Comparing Montgomery Multiplication Algorithms. 1996.
37. A. Martin, M. Burrows, M. Manasse, and T. Wobber. Moderately Hard, Memory-Bound Functions. In *ACM Transcations on Internet Technologies*, 2005.
38. A. May. Secret Exponent Attacks on RSA-type Schemes with Moduli $N = p^r q$. In *Practice and Theory in Public Key Cryptography (PKC)*, 2004.
39. S.F. McGinn and R.E. Shaw. Parallel Gaussian elimination using OpenMP and MPI. In *Proceedings of the International Symposium on High Performance Computing Systems and Applications*, 2002.
40. G. L. Miller. Riemann's Hypothesis and Tests for Primality. In *Proc. Seventh Annual ACM Symp. on the Theory of Computing.*, 1975.
41. M. K. Reiter, V. Sekar, C. Spensky, and Z. Zhang. Making peer-assisted content distribution robust to collusion using bandwidth puzzles. In *International Conference on Information Systems Security*, 2009.
42. R. L. Rivest, A. Shamir, and L. M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. In *Communications of the ACM*, pages 120–126, 1978.
43. R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock Puzzles and Timed-release Crypto. In *MIT Technical Report*, 1996.
44. R. Sedgewick and A. Chi-Chih Yao. Towards Uncheatable Benchmarks. In *Proceedings of The Structure in Complexity Theory Conference*, 1993.
45. S. Tritilanunt, C. Boyd, J. M. Gonzalez Nieto, and E. Foo. Toward Non-Parallelizable Client Puzzles. In *Proceedings of CANS*, 2007.
46. M. van Dijk, D. Clarke, B. Gassend, G. E. Suh, and S. Devadas. Speeding up Exponentiation using an Untrusted Computational Resource. In *Designs, Codes and Cryptography*, volume 39, pages 253–273, 2006.
47. X. Wang and M. Reiter. Defending Against Denial-of-Service Attacks with Puzzle Auctions. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2003.
48. X. Wang and M. K. Reiter. A Multi-layer Framework for Puzzle-based Denial-of-Service Defense. In *International Journal of Information Security*, 2007.
49. B. Waters, A. Juels, J. A. Halderman, and E. W. Felten. New client puzzle outsourcing techniques for DoS resistance. In *Proceedings of the ACM conference on Computer and Communications Security*, 2004.
50. M. Wiener. Cryptanalysis of short RSA secret exponents. In *IEEE Transactions on Information Theory*, pages 553–558, 1990.

A Cryptanalysis of RSA with Large Public Key and Small Private Exponent

Consider an RSA system (N, e, d), where $N = pq$, p and q are large primes, and $e \in \mathbb{N}^+$ such that $e \geq N^2$, $\gcd(e, \phi(N)) = 1$ and d is small. Recall that in RSA, $e \cdot d - 1 = k \cdot \phi(N)$, where $\phi(N) = (p-1)(q-1)$ and $k \in \mathbb{N}^+$.

A.1 Resilience to the Continued Fraction Attack

Theorem 2. *Let $a, b, c, d \in \mathbb{N}^+$ such that $\left|\frac{a}{b} - \frac{c}{d}\right| < \frac{1}{2d^2}$, where $\gcd(a, b) = 1$ and $\gcd(c, d) = 1$ Then, $\frac{c}{d}$ is one of the convergents in the continued fraction expansion of $\frac{a}{b}$. Furthermore, the continued fraction expansion of $\frac{a}{b}$ is finite with the total number of convergents that is polynomial in $\log(b)$.*

In [50], Wiener describes a cryptanalytic attack on the use of an RSA private key $d < N^{0.25}$, when $e < pq$. The attack makes use of an algorithm based on continued fractions that finds the numerator and denominator of a fraction in polynomial time when a close enough estimate of the fraction is known. This will enable the retrieval of a multiple of $\phi(N)$, which will equally result in the factoring of N [40]. The convergence of the continued fraction algorithm is guaranteed when $kd < \frac{pq}{\frac{3}{2}(p+q)}$.

When $e \geq N^2$, $k \geq dpq$. Substituting $k = dpq$ in the equation above yields $d < 1$. More generally, when $e > N^{1.5}$, Wiener's attack will fail since the continued fraction algorithm will not work for any size of the secret exponent d [50].

A.2 Resilience to the Lattice-based Attack

Boneh and Durfee [10] describe a scheme that solves the RSA small-inverse problem when $e < N^{\delta}$ and $d < N^{\alpha}$. As shown in [10], this attack is a heuristic that applies Coppersmith's techniques [15] to bivariate modular polynomials and can only succeed when $\alpha < \frac{7}{6} - \frac{1}{3}\sqrt{1 + 6\delta}$.

Indeed, when $\delta = 1$, $e \leq N$, we achieve the bounds reported in [10]: RSA is insecure when $d < N^{0.292}$. However, when $e \geq N^2$, $\delta > 2$, then this attack will definitely fail ($\alpha < -0.0.35$).

B Cryptanalysis of $ex + y \equiv 0 \mod \phi(N)$

B.1 Single Instance of $ex + y \equiv 0 \mod \phi(N)$

In [8], Blömer *et al.* describe a cryptanalytic attack (based on Wiener's continued fraction algorithm [50]) on a generic RSA key equation of the form $ex + y \equiv 0 \mod \phi(N)$, when $e \leq N, 0 < x < \frac{1}{3}N^{\frac{1}{4}}$ and $|y| < cN^{\frac{-3}{4}}ex$, where $c \leq 1$.

Let $ex + y = k\phi(N) = k(N - p - q + 1)$, where $k \in \mathbb{N}^+$. It then follows that:

$$\frac{e}{N} - \frac{k}{x} = \frac{-y - k(p + q - 1)}{Nx}.$$

The main intuition behind the attack in [8] is to estimate $\frac{k}{x}$ from $\frac{e}{N}$ using the continued fraction algorithm. For the attack to be successful, $\frac{k}{x}$ has to be one of the convergents of $\frac{e}{N}$. This is the case when $|\frac{e}{N} - \frac{k}{x}| = |\frac{-y-k(p+q-1)}{Nx}| < \frac{1}{2x^2}$ (see Theorem 2); that is, when $k(p+q-1)+y < \frac{N}{2x}$.

When $e \geq N^2$, $k \geq N\phi(N)$ ($|\phi(N)| \approx |N|$). It is easy to see in this case that the continued fraction algorithm will not converge ($k(p+q-1)+y \gg \frac{N}{2x}$).

B.2 Multiple Instances of $ex + y \equiv 0 \mod \phi(N)$ with Common Modulus)

Gao (described in [30]) and Howgrave-Graham and Seifert [30] extended Wiener's attack to factor the common modulus when several instances of RSA with $e \leq N$ and $d < N^{0.4-\epsilon}$ are given.

In what follows, we show that these attacks are defeated given several common modulus instances of $ex + y \equiv 0 \mod \phi(N)$ with $e \geq N^2$.

Let $(N_1, e_1), (N_2, e_2)$, be two instances of RSA, then there exists $k_1, k_2 \in \mathbb{N}^+$ such that:

$$e_1 x_1 = -y_1 + k_1 \phi(N)$$
$$e_2 x_2 = -y_2 + k_2 \phi(N)$$

Guo's main observation is that these equations can be combined to remove $\phi(N)$ as follows $k_2 e_1 x_1 - k_1 e_2 x_2 = k_1 y_2 - k_2 y_1$.

With this equation as a starting point, the attack then proceeds in a similar way as Wiener's continued fraction attack:

$$\frac{e_1}{e_2} - \frac{k_1 x_2}{k_2 x_1} = \frac{k_1 y_2 - k_2 y_1}{e_2 k_2 x_1}$$

Given Theorem 2, this suggests that $\frac{k_1 x_2}{k_2 x_1}$ can be obtained from the continued fraction expansion of $\frac{e_1}{e_2}$ when:

$$\left| \frac{k_1 y_2 - k_2 y_1}{e_2 k_2 x_1} \right| < \frac{1}{2(k_2 x_1)^2}$$
$$2k_2 x_1 |k_1 y_2 - k_2 y_1| < e_2$$

When $e_1 > N^2$ and $e_1 > N e_2$, then $2k_2 x_1 |k_1 y_2 - k_2 y_1| \approx \frac{e_2}{\phi(N)} x_1 (\frac{N e_2}{\phi(N)} y_2 - \frac{e_2}{\phi(N)} y_1) \gg e_2$. The continued fraction algorithm will not converge and this attack with then fail. This attack will fail even when $x_1 = x_2$.

Howgrave-Graham and Seifert's attack [30] combines Wiener's, Boneh's and Guo's attacks to factor N given $r \geq 2$ instances of RSA with common modulus. When $e_i > N^2$ and $e_i > N e_j, \forall i \neq j$, their attack will equally fail given any number of common modulus instances[12].

[12] The complexity of existing attacks on common modulus RSA instances increases exponentially with the number of instances; these are only practical for a small number of instances [30], [27].