Practical Client Puzzles in the Standard Model

Lakshmi Kuppusamy Jothi Rangasamy Douglas Stebila

Colin Boyd Juan González Nieto

Information Security Institute, Queensland University of Technology,
GPO Box 2434, Brisbane QLD 4001, Australia
{l.kuppusamy,j.rangasamy,stebila,c.boyd,j.gonzaleznieto}@qut.edu.au

ABSTRACT

Client puzzles are cryptographic problems that are neither easy nor hard to solve. Most puzzles are based on either number theoretic or hash inversions problems. Hash-based puzzles are very efficient but so far have been shown secure only in the random oracle model; number theoretic puzzles, while secure in the standard model, tend to be inefficient. In this paper, we solve the problem of constructing cryptographic puzzles that are secure in the standard model and are very efficient. We present an efficient number theoretic puzzle that satisfies the puzzle security definition of Chen *et al.* (ASIACRYPT 2009). To prove the security of our puzzle, we introduce a new variant of the interval discrete logarithm assumption which may be of independent interest, and show this new problem to be hard under reasonable assumptions. Our experimental results show that, for 512-bit modulus, the solution verification time of our proposed puzzle can be up to 50× and 89× faster than the Karame-Čapkun puzzle and the Rivest *et al.*'s time-lock puzzle respectively. In particular, the solution verification time of our puzzle is only 1.4× slower than that of Chen *et al.*'s efficient hash based puzzle.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Client/Server*; D.4.6 [**Operating Systems**]: Security and Protection—*Authentication*

General Terms

Security

Keywords

client puzzle, denial of service, interval discrete log problem, factorisation, puzzle unforgeability, puzzle difficulty

ASIACCS '12, May 2–4, 2012, Seoul, Korea.

1. INTRODUCTION

Denial-of-service (DoS) attacks are a growing concern to networked services like the Internet. In recent years, major Internet e-commerce and government sites have been disabled due to various DoS attacks. A common form of DoS attack is a *resource depletion attack*, in which an attacker tries to overload the server's resources, such as memory or computational power, rendering the server unable to service honest clients. A promising way to deal with this problem is for a defending server to identify and segregate malicious traffic as earlier as possible.

Client puzzles, also known as *proofs of work*, have been shown to be a promising tool to thwart DoS attacks in network protocols, particularly in authentication protocols. A puzzle is issued by the server in reply to each request when the server is under attack. After receiving a puzzle, the client has to solve it in order to convince the server to allocate its resources. The main idea is that puzzle generation and solution verification should be easy for the server, while computing the puzzle solution should be somewhat computationally hard for the client.

Many client puzzles have been proposed since they were first introduced by Dwork and Naor in 1992 [7]. An important recent development has been the analysis of client puzzles in the provable security framework [6, 19]. The computational problems underlying most puzzles are either number-theoretic [7, 12, 21] or based on hash inversions [3, 6, 10, 11]. Hash-based puzzles are very efficient — generation and verification typically requires only one or two hash function calls — but concrete realisations to date have been shown secure only in the random oracle model. Number-theoretic puzzles, on the other hand, have been shown secure in the standard model but have tended to be relatively inefficient, typically requiring the server to perform a large integer modular exponentiation making it unsuitable for high speed applications. Recently, Rangasamy *et al.* [16] proposed an efficient modular exponentiation based client puzzle which does not require any online exponentiations for puzzle generation and verification. But the puzzle security relies on the security of time-lock puzzle of Rivest *et al.* [17] and does not follow from standard model security assumptions. The existence of a highly efficient, standard-model secure client puzzle has remained an open question until now.

1.1 Contributions

Our contributions in this work are as follows:

- We propose an efficient number-theoretic client puzzle

Puzzle	Standard model	Generation Cost	Generation Time (μs)	Verification Cost	verification Time (μs)
Rivest *et al.* [17]	Yes	1 hash	4.80	1 hash $\|n\|$-bit mod. exp.	474.68
Karame-Čapkun [12]	Yes	2 HMAC (4 hash) 1 gcd	8.37	2 HMAC (4 hash) $2k$-bit mod. exp.	263.35
Chen *et al.* hash based puz [6]	No	1 HMAC and 1 hash	5.92	1 HMAC 1 hash	3.77
Rangasamy *et al.* [16]	Yes	1 hash $2(\ell-1)$ mod. mul.	16.66	1 hash 3 mod. mul.	14.75
Our Puzzle	Yes	1 HMAC, $(\ell-1)$ mod. add. $(\ell-1)$ mod. mul. 1 large integer mul. 1 large integer add.	31.43	1 HMAC 1 large integer mul. 1 large integer add.	5.31

Table 1: Puzzle generation and verification costs and time for low difficulty level Q, 512-bit RSA modulus n, 56-bit security parameter k. ℓ is set to 4 and 8 for Rangasamy *et al.*'s puzzle and our puzzle respectively.

DLPuz that does not require any online exponentiations. In fact, our puzzle requires only a few modular additions and multiplications for generation and verification which is a significant improvement over the existing practical standard model puzzles. Thus we solve the problem of constructing cryptographic puzzles that are secure in the standard model and are very efficient. Table 1 lists the generation and verification costs and times for our puzzle and other puzzles.

- We compare the performance of our puzzle with the performances of the puzzles listed in Table 1. Our experimental results show that the solution verification time of our puzzle is approximately 89 times faster when compared to Rivest *et al.*'s time-lock puzzle and by approximately 50 times faster when compared to Karame and Čapkun, for 512-bit RSA moduli. The solution verification time of our puzzle is approximately 3 times faster than that of Rangasamy *et al.*'s efficient number theoretic puzzle. The puzzle verification time of our puzzle is only 1.4 times slower than Chen *et al.*'s most efficient hash based puzzle which is proven secure in the random oracle model.

- We analyse the security of our puzzle using the puzzle security model of Chen *et al.* [6] and show that our puzzle satisfies the unforgeability and difficulty properties.

- Though our puzzle enjoys a simple construction, its security does not follow directly from existing cryptographic assumptions. We introduce a new variant of the composite interval discrete logarithm assumption which we call the *modular composite interval discrete logarithm assumption*, IDL*. We show that this new computational problem is as hard as the composite interval discrete logarithm problem (IDL) and the factorisation problem.

Outline.

The rest of the paper is organised as follows. Section 2 presents the background and motivation for our work. In Section 3, we present our new client puzzle scheme DLPuz. In Section 4, we introduce a new variant IDL* of the composite interval discrete logarithm assumption and analyse its hardness. Section 5 describes the puzzle security model of Chen *et al.* and provides the security analysis of DLPuz in the standard model, relating its difficulty to the new IDL* problem. Finally, we present our experimental results in Section 6 and conclude the paper with future work in Section 7.

2. BACKGROUND

In this section, we review the relevant literature on client puzzles, with an emphasis on standard model puzzles.

Client Puzzles. Client puzzles were first proposed by Dwork and Naor [7] as a countermeasure for email spam. Many client puzzles [3, 6, 10, 11] are based on the difficulty of inverting a hash function. These hash-based puzzles are generally quite efficient: typically they require only one or two hash function calls for puzzle generation and verification. In this work we will focus on number theoretic puzzles, and will review the various constructions below.

Until recently, the difficulty of solving puzzles was addressed in a mostly *ad hoc* manner. However, several provable security models have been recently introduced: one by Chen *et al.* [6] for analysing the difficulty of solving a single puzzle, and one by Stebila *et al.* [19] for the case of solving multiple puzzles.

Modular Exponentiation-Based Puzzles. In 1996, Rivest *et al.* [17] introduced *time-lock puzzles* that can *only* be solved by running a computer continuously for a certain amount of time. An example construction of a time-lock puzzle was given based on repeated squaring. Given a composite RSA modulus n and a random element a in $\mathbb{Z}_n^*$, the client's task is to do t repeated squaring on a: compute $a^{2^t} \mod n$.

The server can use its knowledge of $\phi(n)$ as a shortcut to create the puzzle with two modular exponentiations, whereas the client has to spend no less than the predetermined amount of time to solve the puzzle. However, because of the high cost of puzzle generation and verification at the server, time-lock puzzles are not suitable for DoS defense.

In 2010, Karame and Čapkun [12] reduced the verification cost of the time-lock puzzles of Rivest *et al.*. The puzzle scheme works as follows: Let (n, e, d) be a tuple of RSA parameters such that $e \cdot d = 1 \mod \phi(n)$. Here d is chosen to be small such that $d \geq k$, where k is a security parameter. Instead of an RSA public key e, an enlarged public key $\bar{e}$ is given as the puzzle and a client needs to encrypt a challenge string. The difficulty of a puzzle is adjusted by increasing or decreasing the size of $\bar{e}$. The server verifies a solution by decrypting with d (which is small) and checking whether the resulting value matches the challenge.

The speed-up achieved was a factor of $\frac{\log n}{k}$, when $\log d = k+1$. For example, when a 1024-bit modulus is used, the full 1024-bit modular exponentiation required for solution verification in time-lock puzzles is reduced to a 1024-bit modular exponentiation with an 80-bit exponent. Although, this improvement is significant compared with the performance of known time-lock puzzles, the puzzle is still not suitable for high-speed practical applications because of its higher verification cost.

Recently, Rangasamy *et al.* [16] proposed a modular exponentiation-based client puzzle which can be seen as an efficient alternative to Rivest *et al.*'s time-lock puzzle. Unlike the Rivest *et al.*and Karame-Čapkun puzzle, Rangasamy *et al.*'s puzzle does not require the server to perform any online exponentiations. In fact, the server has to perform totally two hash operations and few modular multiplications for the puzzle generation and verification. Although it is a significant improvement over the Karame-Čapkun puzzle construction, the security of the puzzle does not rely on the standard security assumptions.

Diffie-Hellman-Based Puzzles. Waters *et al.* [21] proposed a puzzle based on the Diffie-Hellman (DH) problem. Given a generator β of a group of prime order q and a random integer a in $[r, r+Q]$, a puzzle consists of the values $(\beta^{f(a)}, r, Q)$, where f is a one-way permutation on $\mathbb{Z}_q$ and Q is a difficulty parameter. The client solves the puzzle by checking each candidate value $\tilde{a} \in [r, r+Q]$ to see if $\beta^{f(\tilde{a})} = \beta^{f(a)}$. By giving a hint interval $[r, r+Q]$ to the client, the puzzle difficulty achieves *linear granularity.*

To contact a particular server, the client needs to do extra work by combining the puzzle solution with the server's DH public key; that is, the client actually calculates $(\beta^x)^{f(a)}$ as $y^{f(a)}$, where y is the public key of the server. The server needs one modular exponentiation to verify the solution, by raising the puzzle to its private key x: $(\beta^{f(a)})^x$. Since the defending server can independently compute the solution for a time period, all the solutions for the particular time period are precomputed and stored by the server so that verification needs a single table lookup.

While verification via table lookup is considered to be cheap, constructing a puzzle still requires one modular exponentiation which is expensive and thus is undesirable for a defending server. To avoid this circumstance, Waters *et al.* suggested outsourcing the puzzle creation to a secure third party, called a *bastion*, thereby removing the computational burden of puzzle generation from the server. However, the assumption on existence of such a third party seems to be unsatisfactory.

3. DLPuz: AN EFFICIENT NUMBER-THEORETIC PUZZLE

This section describes our new client puzzle construction DLPuz, which is based on the problem of finding a discrete logarithm in an interval. First we review the definition of a client puzzle and then present our construction.

Notation. If n is an integer, then we use $|n|$ to denote the length in bits of n, and $\phi(n)$ is the Euler phi function for n. If S is a set, then $x \leftarrow_R S$ denotes choosing x uniformly at random from S. If $\mathcal{A}$ is an algorithm, then $x \leftarrow \mathcal{A}(y)$ denotes assigning to x the output of $\mathcal{A}$ when run with the input y. An interval of integers is denoted by $[a, b]$. If $\mathcal{I}$ is an interval, we note in particular that $\mathcal{I} \leftarrow [a, b]$ denotes setting $\mathcal{I}$ to be the interval with endpoints a and b, *not* selecting an element from that interval. If k is a security parameter, then $\mathrm{negl}(k)$ denotes a function that is negligible in k, namely asymptotically smaller than the inverse of any polynomial in k.

3.1 Defining Client Puzzles

Chen *et al.* [6] gave the following definition of a client puzzle:

DEFINITION 1. *Client Puzzle A* client puzzle Puz *is a tuple consisting of the following algorithms:*

- $\mathsf{Setup}(1^k)$*: A p.p.t. setup algorithm that generates and returns a set of public parameters params and a secret key s, the former of which includes a puzzle difficulty parameter space* QSpace*.*
- $\mathsf{GenPuz}(s, Q, str)$*: A p.p.t. puzzle generation algorithm which accepts a secret key s, difficulty parameter Q, and a session string str and returns a puzzle puz.*
- $\mathsf{FindSoln}(puz, t)$*: A probabilistic puzzle solving algorithm that returns a potential solution soln for puzzle puz after running time at most t.*
- $\mathsf{VerAuth}(s, puz)$*: A d.p.t. puzzle authenticity verification algorithm that returns* true *or* false
- $\mathsf{VerSoln}(s, str, puz, soln)$*: A d.p.t. puzzle solution verification algorithm that returns* true *or* false*.*

For *correctness*, we require that if $(params, s) \leftarrow \mathsf{Setup}(1^k)$ and $puz \leftarrow \mathsf{GenPuz}(s, Q, str)$ then there exists $t \in \mathbb{N}$ such that $\mathsf{VerSoln}(s, str, puz, soln)$ is true with probability 1 where $soln \leftarrow \mathsf{FindSoln}(puz, t)$.

3.2 The DLPuz Puzzle

The idea behind our proposed puzzle scheme is the following: given a RSA modulus n, $g, V = g^v \mod n$ and an interval $\mathcal{I}$, where $v \in \mathcal{I}$, the task of a client is to find v.

Waters *et al.* [21] outsourced the computation of $g^v \mod n$ to a trusted third party, thereby removing the computational burden associated with the puzzle generation. In this work, we show how to shift this burden to clients while maintaining the secrecy of the solution. Hence, we do not assume the existence of such a trusted third party, thereby making our proposal more practical.

Our puzzle construction makes use of several other cryptographic primitives. It relies on a modulus generation algorithm GenRSA that generates an RSA-style modulus $n = pq$. We note that RSA modulus generation only needs to be done once in the Setup phase, not in each puzzle generation. Our puzzle also employs a technique due to Boyko *et al.* [5] for quickly generating many ephemeral values g^x using a relatively small amount of precomputation.

DEFINITION 2. *(Modulus Generation Algorithm) Let k be a security parameter. A* modulus generation algorithm *is a probabilistic polynomial-time algorithm* GenRSA *that, on input* 1^k*, outputs* (n, p, q) *such that* $n = pq$ *and* p *and* q *are k-bit primes.*

In our puzzle generation algorithm, the server has to generate a pair (a, g^a) for each puzzle. Since the generation of these pairs are expensive, the server uses the following generator proposed by Boyko *et al.* [5] to efficiently generate such pairs for each puzzle.

DEFINITION 3. *(BPV Generator) Let k, ℓ, and N, with $N \geq \ell \geq 1$, be parameters. Let $n \leftarrow \mathsf{GenRSA}(1^k)$ be an RSA modulus. Let g be a random element of order M in the multiplicative group $\mathbb{Z}_n^*$. A* BPV generator *consists of the following two algorithms:*

- $\mathsf{BPVPre}(g, n, N, M)$*: A pre-processing algorithm that is run once. Generate N random integers $x_1, x_2, \ldots, x_N \leftarrow_R \mathbb{Z}_M$. Compute $X_i \leftarrow g^{x_i} \mod n$ for each i. Return a table $\tau \leftarrow ((x_i, X_i))_{i=1}^N$.*
- $\mathsf{BPVGen}(g, n, \ell, M, \tau)$*: A pair generation algorithm that is run whenever a pair (a, g^a) is needed. Choose a random set $S \subseteq_R \{1, \ldots, N\}$ of size ℓ. Compute $u \leftarrow \sum_{j \in S} x_j \mod M$. If $u = 0$, then stop and generate S again. Otherwise, compute $U \leftarrow \prod_{j \in S} g^{x_j} \mod n$ and return (u, U). In particular, the indices S and the corresponding pairs $((x_j, X_j))_{j \in S}$ are not revealed.*

Randomness of BPV generator.

Boyko *et al.* [5] proposed the discrete-log-based BPV generator to speed up protocols based on discrete logarithm such as Elgamal, DSS and Schnorr signatures, Diffie-Hellman key exchange, and Elgamal encryption. In Boyko's thesis [4], it is proved (Claim 4) that the outputs of the BPV generator are statistically indistinguishable from uniform values for large values of $\binom{N}{\ell}$. Nguyen *et al.* [13] proposed the extended BPV generator (EBPV) and argued that for the BPV generator is the special case of EBPV generator and hence the security results for EBPV also holds for BPV. They established the security of some discrete logarithm based signature schemes that use EBPV under adaptive chosen message attack. They also obtained results for the statistical distance between the distribution of EBPV and the uniform distribution.

Nguyen and Stern [14] analysed the distribution of the output of the BPV generator and showed in Theorem 1 that for a fixed M, with overwhelming probability on the choice of x_i's, the distribution of the BPV generator is statistically close to the uniform distribution [14]. In particular, a polynomial time adversary cannot distinguish the two distributions

THEOREM 1. *For all $M > 0$, if $x_1, \ldots x_N$ are chosen independently and uniformly from $[0, M-1]$ and if $a = \sum_{j \in S} x_j \mod M$ is computed from a random set $S \subseteq \{1, \ldots N\}$ of ℓ elements, then the statistical distance between the computed a and a randomly chosen $a' \in Z_M$ is bounded by $\sqrt{M/\binom{N}{\ell}}$. That is,*

$$\left| \Pr\left(\sum_{j \in S} x_j = a \mod M \right) - \frac{1}{M} \right| \leq \sqrt{M/\binom{N}{\ell}}$$

The above theorem is valid even if one considers polynomially many samples, not just one. The bound for BPV is shown in Theorem 1. Moreover, if the statistical difference of some distributions $\mathcal{D}_1$ and $\mathcal{D}_2$ defined over a set $\mathcal{S}$ is less than ϵ, then the statistical difference of $\mathcal{D}_1^u$ and $\mathcal{D}_2^u$ defined over the set $\mathcal{S}^u$ is less than $u\epsilon$, where $\mathcal{D}_1^u$ is defined by choosing m elements independently random from $\mathcal{S}$.

3.3 Definition of DLPuz

We now give the definition of our puzzle DLPuz in Figure 1, which we have organised diagrammatically to suggest an interaction between a server issuing puzzles and a client solving them. DLPuz is parameterised by a security parameter k, a difficulty parameter Q. In practice, a server using client puzzles as a denial-of-service countermeasure can vary Q based on the severity of the attack it is experiencing.

Puzzle solving. One typical method for a legitimate client to implement the FindSoln algorithm is a brute-force search. Upon receiving a puzzle *puz* from the server with an interval $[i, i+Q]$, the client computes V and $g^i \mod n$. It then iterates by multiplying the current value with $g \mod n$ and comparing that value with V. If the length of the interval $\mathcal{I}$ is Q, then this will take approximately $Q/2$ multiplications on average, plus the cost of the initial exponentiations. We note however that a client could also choose to solve this problem using one of the faster interval-kangaroo techniques described by Galbraith *et al.* [8] which require approximately $O(\sqrt{Q})$ steps plus the cost of the initial exponentiations.

Server efficiency. In many scenarios, it is essential that the GenPuz, VerAuth, and VerSoln algorithms be extremely efficient. In a denial-of-service setting, these algorithms are run online by the server many times, and if they were expensive then an attacker could induce a resource depletion attack by asking for many puzzles to be generated or verified.

GenPuz: The dominant cost in puzzle generation is the BPV pair generation BPVGen, which requires $\ell - 1$ modular additions and $\ell - 1$ modular multiplications. There is also a single call to the HMAC H_ρ (a keyed collision-reisistant pseudo-random function where ρ is used as a key), a large integer multiplication $b \cdot z$, and three integer additions.

VerAuth: Puzzle authenticity verification is quite cheap, requiring just a single call to the HMAC H_ρ.

VerSoln: To verify correctness of a solution, the server has to perform only 1 modular addition and 1 modular multiplication.

REMARK 1. *In* DLPuz *as specified in Figure 1, the server has to store a short-term secret a to re-compute v for verifying the solution. If the server stores a for each puzzle, then it may be vulnerable to a memory-based DoS attack. To avoid this type of attack, the server may use a* stateless connection *[1] to offload storage of a to the client. In particular, the server can encrypt a under a long-term symmetric key sk and send it along with each puzzle. Then the client has to echo it back while sending the solution to the puzzle. In this way, the server remains stateless and can obtain a by decrypting the encrypted value using the key sk. The cost for encryption and decryption adds a very little cost to the server. For example, the time to encrypt or decrypt 512 bits of data using AES-128-CBC is approximately 0.403 micro seconds and hence we ignore these costs in the performance comparison section.*

3.4 Parameter Sizes for DLPuz

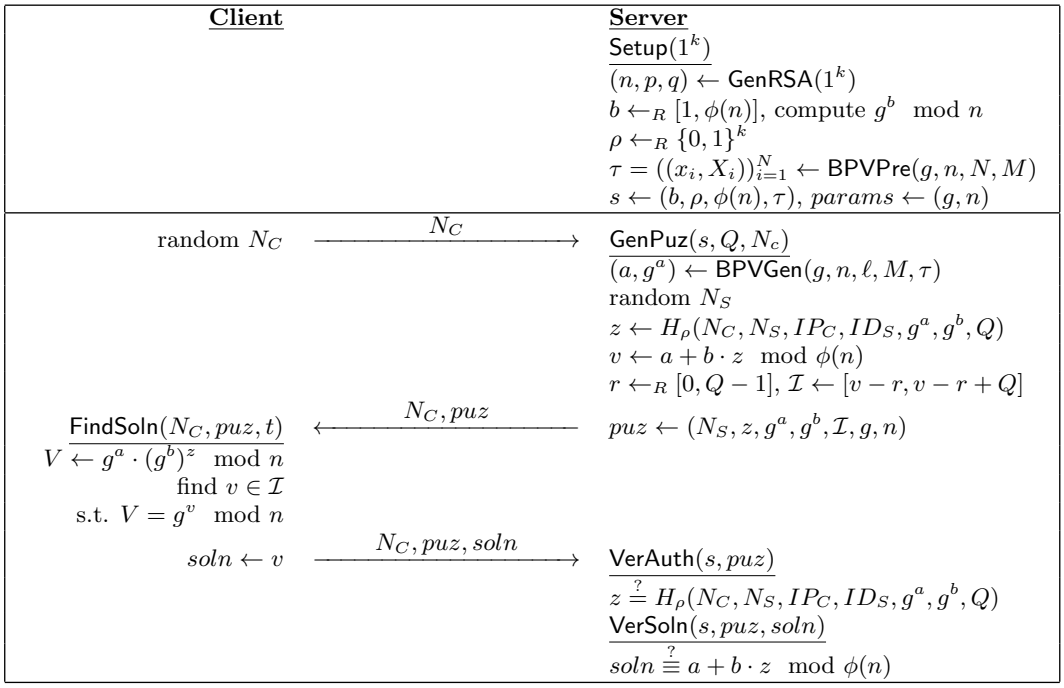

Figure 1: Client puzzle DLPuz based on the interval discrete logarithm problem

Number of pairs to pre-compute	Values of $\sqrt{M/\binom{N}{\ell}}$ for a 80-bit M and the chosen ℓ									
N	$\ell=8$	$\ell=9$	$\ell=10$	$\ell=11$	$\ell=12$	$\ell=13$	$\ell=14$	$\ell=15$	$\ell=16$	$\ell=17$
2^{10}	2^{4}	2^{-1}	2^{-5}	2^{-9}	2^{-14}	2^{-18}	2^{-23}	2^{-27}	2^{-32}	2^{-36}
2^{11}	2	2^{-5}	2^{-10}	2^{-15}	2^{-20}	2^{-25}	2^{-30}	2^{-35}	2^{-40}	2^{-45}
2^{12}	2^{-4}	2^{-9}	2^{-15}	2^{-20}	2^{-26}	2^{-31}	2^{-37}	2^{-42}	2^{-48}	2^{-53}
2^{13}	2^{-8}	2^{-14}	2^{-20}	2^{-26}	2^{-32}	2^{-38}	2^{-44}	2^{-50}	2^{-56}	2^{-62}
2^{14}	2^{-12}	2^{-18}	2^{-25}	2^{-30}	2^{-38}	2^{-44}	2^{-51}	2^{-57}	2^{-64}	2^{-70}
2^{15}	2^{-16}	2^{-23}	2^{-30}	2^{-37}	2^{-44}	2^{-51}	2^{-58}	2^{-65}	2^{-72}	2^{-79}
2^{16}	2^{-20}	2^{-27}	2^{-35}	2^{-42}	2^{-50}	2^{-57}	2^{-65}	2^{-72}	2^{-80}	2^{-87}
2^{17}	2^{-24}	2^{-32}	2^{-40}	2^{-48}	2^{-56}	2^{-64}	2^{-72}	2^{-80}	2^{-88}	2^{-96}

Table 2: Distinguishability of BPV Pairs from Random pairs

Our DLPuz requires a pair (a, g^a) to be computed during each puzzle generation. We use BPV generator to efficiently generate such a pair using the N pre-computed pairs (x_i, X_i). The efficiency of puzzle generation depends on the number of elements ℓ in the random set S the server choose to compute (a, g^a). Note that a defending DoS server may prefer to reduce the number of modular multiplications required for each puzzle generation. Hence it might be appropriate for the server to choose the bigger value of N (polynomial in $\log M$) to make ℓ smaller. Table 2 specifies the approximate distribution distance between pairs (a, g^a) generated uniformly at random versus pairs generated by the BPV generator using Theorem 1 for the specified N and ℓ values, with a 80-bit M value. An example showing the statistical distance for specific parameter values appear in Appendix A.

4. A NEW VARIANT OF THE INTERVAL DISCRETE LOGARITHM PROBLEM

Computing discrete logarithms in an interval is a fundamental computational problem that has arisen naturally in a number of contexts [9, 15, 20]. The security of our DLPuz puzzle relies on a variant IDL^* of the Interval Discrete Log (IDL) assumption; the main difference in our variant is that the adversary can return any value x' which is equivalent, modulo $\phi(n)$ (where n is an RSA modulus), to the discrete logarithm of g^x. In this section, we introduce a new variant IDL^* and show that our new IDL^* problem is as hard as the original IDL problem and integer factorisation. The formal definition of the factorisation and the interval discrete log problem specifically for the RSA composite modulus n appear in Appendix B.

4.1 The Modular Composite Interval Discrete Logarithm Problem

We now describe our variant of the IDL problem. Given a modulus $n = pq$, an element $y = g^x \mod n$, and an interval $\mathcal{I}$ of length q such that $x \in \mathcal{I}$, the *modular composite interval discrete logarithm problem* IDL^* is to compute x' such that $x \equiv x' \mod \phi(n)$.

Definition 4. *(Modular Composite Interval Discrete Logarithm Problem IDL^*) Let k be a security parameter, q be a difficulty parameter, and GenRSA be a modulus generation algorithm. Let $\mathcal{A}$ be a probabilistic algorithm. Define the experiment $\mathsf{Exp}^{\mathsf{IDL}^*}_{\mathcal{A},\mathsf{GenRSA},q}(k)$ as follows:*

1. $n \leftarrow \mathsf{GenRSA}(1^k)$.
2. $g \leftarrow_R \mathbb{Z}_n^*$, $x \leftarrow_R [1, \phi(n)]$, $y \leftarrow g^x \mod n$.
3. $r \leftarrow_R [0, q-1]$, $\mathcal{I} \leftarrow [x - r, x - r + q]$.
4. $x' \leftarrow \mathcal{A}(g, y, n, \mathcal{I})$.
5. *Output* 1 *if* $x' \equiv x \mod \phi(n)$ *and* 0 *otherwise.*

The advantage of $\mathcal{A}$ in violating the IDL^* assumption is

$$\mathsf{Adv}^{\mathsf{IDL}^*}_{\mathcal{A},\mathsf{GenRSA},q}(k) = \Pr\left(\mathsf{Exp}^{\mathsf{IDL}^*}_{\mathcal{A},\mathsf{GenRSA},q}(k) = 1\right) .$$

The IDL^* problem with GenRSA is said to be $\delta_{k,q}(t)$-hard if $\mathsf{Adv}^{\mathsf{IDL}^*}_{\mathcal{A},\mathsf{GenRSA},q}(k) \leq \delta_{k,q}(t)$ for all $\mathcal{A}$ running in time at most t.

In the following theorem we show that solving the IDL^* problem is as hard as solving either the IDL problem or the integer factorisation problem.

Theorem 2 (Hardness of IDL^*). *Let k be a security parameter, q be a difficulty parameter, and GenRSA be a modulus generation algorithm. Suppose there exists a probabilistic algorithm $\mathcal{A}$ running in time t which can solve the IDL^* problem for GenRSA on an interval of size q. Then there exists a probabilistic algorithm $\mathcal{B}$ with running time $t' = t + t_{\mathsf{exp}}(k) + c$, where $t_{\mathsf{exp}}(k)$ is the time to compute an exponentiation modulo an output of $\mathsf{GenRSA}(1^k)$ and c is a constant, that solves either the factorisation problem for GenRSA or the IDL problem on an interval of size q. In particular,*

$$\mathsf{Adv}^{\mathsf{IDL}^*}_{\mathcal{A},\mathsf{GenRSA},q}(k) \leq \mathsf{Adv}^{\mathsf{Fact}}_{\mathcal{B},\mathsf{GenRSA}}(k) + \mathsf{Adv}^{\mathsf{IDL}}_{\mathcal{B},\mathsf{GenRSA},q}(k) + \mathrm{negl}(k) .$$

Proof. Let $\mathcal{A}$ be a probabilistic algorithm with running time t. We prove the theorem using a sequence of games [18]. In one of the games, we will insert a factorisation challenge and a win by the adversary lets us factor; in another game, we will insert an IDL challenge and a win by the adversary gives us the discrete logarithm.

Let S_i be the event the adversary $\mathcal{A}$ wins game G_i.

Game G_0.

Let G_0 be the original IDL^* experiment. Thus,

$$\mathsf{Adv}^{\mathsf{IDL}^*}_{\mathcal{A},\mathsf{GenRSA},q}(k) = \Pr(\mathsf{S}_0) . \quad (1)$$

Game G_1.

In game G_1, the challenger chooses x from the interval $[1, n]$ instead of the interval $[1, \phi(n)]$. Since $\phi(n)$ is very close to n, the distribution of messages returned by the challenger is virtually unchanged. In particular, the probability that x lies in an interval $[\phi(n), n]$ is $(n - \phi(n))/n = O(1/\sqrt{n})$ which is negligible in k, and hence

$$|\Pr(\mathsf{S}_0) - \Pr(\mathsf{S}_1)| \leq O\left(1/\sqrt{n}\right) \leq \mathrm{negl}(k) . \quad (2)$$

Game G_2: Factorisation.

The change from G_1 to G_2 is that in G_2 the challenger uses the adversary to try to factor a modulus n from a factorisation challenger by simulating the IDL^* experiment from game G_1.

To begin, the IDL^* challenger $\mathcal{B}$ obtains a factorisation challenge: it is given n such that $n = pq$ and must compute either p or q. With this n, $\mathcal{B}$ chooses the values g, x, y, r, and $\mathcal{I}$ as in game G_1. $\mathcal{B}$ then initiates $\mathcal{A}$ with the inputs $(g, y, n, \mathcal{I})$. Suppose $\mathcal{A}$ solves the IDL^* problem. Let y be the element output by $\mathcal{A}$; we have that $y \equiv x \mod \phi(n)$. Here either $y = x$ or $y = x + m\phi(n)$. Let F be the event that $y = x + m\phi(n)$ where $m \geq 1$.

When F occurs, $\mathcal{B}$ can compute $y - x = m\phi(n)$. With a multiple of $\phi(n)$, $\mathcal{B}$ can now compute a non-trivial factor of n. Hence,

$$\Pr(\mathsf{S}_2 | \mathsf{F}) \leq \mathsf{Adv}^{\mathsf{Fact}}_{\mathcal{B},\mathsf{GenRSA}}(k) \quad (3)$$

and the running time of $\mathcal{B}$ is $t(\mathcal{B}) = t(\mathcal{A}) + t_{\mathsf{exp}} + c$, where t_{exp} is the time to perform an exponentiation $g^x \mod n$ and c is a constant. We also note that, since the distribution of values provided by $\mathcal{B}$ to $\mathcal{A}$ are exactly the same as in game

G_1, we have

$$|\Pr(\mathsf{S}_1|\mathsf{F}) - \Pr(\mathsf{S}_2|\mathsf{F})| = 0 \ . \tag{4}$$

When F does not occur, we do not have any way of solving the factorisation problem. However, we will construct game G_3 in which we can solve an IDL challenge when $\bar{\mathsf{F}}$ occurs.

Game G_3: IDL.

Game G_3 is based on G_1 (not G_2); the change from G_1 to G_3 is that in G_3 the challenger uses the adversary to try to solve a modular composite interval discrete logarithm problem from an IDL challenger by simulating the IDL^* experiment from game G_1.

The IDL^* challenger $\mathcal{B}$ obtains an IDL challenge: it is given $(g, g^x, n, \mathcal{I})$, and must compute x. $\mathcal{B}$ passes the inputs $(g, g^x, n, \mathcal{I})$ to $\mathcal{A}$. Suppose $\mathcal{A}$ solves the IDL^* problem. Let y be the element output by $\mathcal{A}$; we have that $y \equiv x \mod \phi(n)$. Here either $y = x$ or $y = x + m\phi(n)$. Again, let F be the event that $y = x + m\phi(n)$ where $m \geq 1$.

When $\bar{\mathsf{F}}$ occurs, $\mathcal{B}$ has a solution $y = x$ to the IDL challenge it was given. Hence,

$$\Pr(\mathsf{S}_3|\bar{\mathsf{F}}) \leq \mathsf{Adv}^{\mathsf{IDL}}_{\mathcal{B},\mathsf{GenRSA},q}(k) \tag{5}$$

and the running time of $\mathcal{B}$ is $t(\mathcal{B}) = t(\mathcal{A})$. We also note that, since the distribution of values provided by $\mathcal{B}$ to $\mathcal{A}$ are exactly the same as in game G_1, we have

$$\left|\Pr(\mathsf{S}_1|\bar{\mathsf{F}}) - \Pr(\mathsf{S}_3|\bar{\mathsf{F}})\right| = 0 \ . \tag{6}$$

When $\bar{\mathsf{F}}$ does not occur, we do not have any way of solving the IDL problem. However, game G_2 handles the event when F occurs.

Analysis of Game G_1.

Combining equations (3)–(6), we find

$$\begin{aligned}\Pr(\mathsf{S}_1) &= \Pr(\mathsf{F})\Pr(\mathsf{S}_1|\mathsf{F}) + \Pr(\bar{\mathsf{F}})\Pr(\mathsf{S}_1|\bar{\mathsf{F}}) \\ &= \Pr(\mathsf{F})\Pr(\mathsf{S}_2|\mathsf{F}) + \Pr(\bar{\mathsf{F}})\Pr(\mathsf{S}_3|\bar{\mathsf{F}}) \\ &\leq \Pr(\mathsf{F})\mathsf{Adv}^{\mathsf{Fact}}_{\mathcal{B},\mathsf{GenRSA}}(k) + \Pr(\bar{\mathsf{F}})\mathsf{Adv}^{\mathsf{IDL}}_{\mathcal{B},\mathsf{GenRSA},q}(k) \\ &\leq \mathsf{Adv}^{\mathsf{Fact}}_{\mathcal{B},\mathsf{GenRSA}}(k) + \mathsf{Adv}^{\mathsf{IDL}}_{\mathcal{B},\mathsf{GenRSA},q}(k) \ . \end{aligned} \tag{7}$$

Final result.

The result follows by combining equations (1), (2), and (7). □

5. SECURITY ANALYSIS OF DLPuz

In this section, we analyse the DLPuz puzzle using the security model of Chen *et al.* [6]. Chen *et al.* introduced two security properties that a client puzzle should satisfy: unforgeability and difficulty. We give a brief description of these two properties. Unforgeability of DLPuz follows from the straightforward use of a pseudo-random function as a message authentication code. We show that the difficulty of DLPuz can be reduced to the IDL^* problem.

5.1 Unforgeability

This experiment measures the ability of an adversary to produce a valid client puzzle and force a server to accept it as one that was not originally generated by a server in a probabilistic way.

In general, unforgeability can easily be provided by using a message authentication code (MAC) or pseudo-random function to tag puzzles generated by the server, and this is what done in DLPuz. The formal definition of puzzle unforgeability and the result showing that DLPuz is indeed unforgeable appear in Appendix C due to space constraints.

5.2 Difficulty

The difficulty property ensures that an adversary has to spend the specified amount of resources to solve an instance of a client puzzle. In the following theorem, we show that our puzzle, DLPuz, is a difficult puzzle under the IDL^* assumption.

DEFINITION 5. *(Puzzle Difficulty [6]) Let k be a security parameter and let Q be a difficulty parameter which is kept fixed through the experiment. Let $\mathcal{A}$ be a probabilistic algorithm and* Puz *be a client puzzle. Define the experiment $\mathsf{Exp}^{\mathsf{Diff}}_{\mathcal{A},\mathsf{Puz},Q}(k)$ as follows:*

1. $(params, s) \leftarrow \mathsf{Setup}(1^k)$.
2. *Run $\mathcal{A}(params)$ with oracle access to* $\mathsf{CreatePuzSoln}(\cdot)$ *and* $\mathsf{Test}(\cdot)$*, which are answered as follows:*
   - $\mathsf{CreatePuzSoln}(str)$*: $puz \leftarrow \mathsf{GenPuz}(s, Q, str)$. Find a solution $soln$ such that $\mathsf{VerSoln}(puz, soln) = \mathsf{true}$. Return $(puz, soln)$ to $\mathcal{A}$.*
   - $\mathsf{Test}(str^*)$*: This query may be asked once, at any point during the game. The challenger generates a puzzle $puz^* \leftarrow \mathsf{GenPuz}(s, Q, str)$ and returns puz^* to $\mathcal{A}$. Then $\mathcal{A}$ may continue to ask* CreatePuzSoln *queries.*
3. *$\mathcal{A}$ outputs a potential solution $soln^*$.*
4. *Output 1 if $\mathsf{VerSoln}(puz^*, soln^*) = \mathsf{true}$ and 0 otherwise.*

We say that $\mathcal{A}$ wins the game if $\mathsf{Exp}^{\mathsf{Diff}}_{\mathcal{A},\mathsf{Puz},Q}(k) = 1$ and loses otherwise. The advantage of $\mathcal{A}$ is defined as:

$$\mathsf{Adv}^{\mathsf{Diff}}_{\mathcal{A},\mathsf{Puz},Q}(k) = \Pr\left(\mathsf{Exp}^{\mathsf{Diff}}_{\mathcal{A},\mathsf{Puz},Q}(k) = 1\right) \ .$$

Let $\epsilon_{k,Q}(t)$ be a family of functions monotonically increasing in t. A puzzle Puz *is $\epsilon_{k,Q}(t)$−*difficult *if, for all probabilistic algorithms $\mathcal{A}$ running in time at most t,*

$$\mathsf{Adv}^{\mathsf{Diff}}_{\mathcal{A},\mathsf{Puz},Q}(k) \leq \epsilon_{k,Q}(t) \ .$$

THEOREM 3 (DIFFICULTY OF DLPuz). *Let k be a security parameter and let Q be a difficulty parameter. Let* GenRSA *be a modulus generation algorithm and let H_ρ be a pseudo-random function. Suppose IDL^* with* GenRSA *is $\delta_{k,Q}(t)$-difficult. Let $\epsilon_{k,Q}(t) = \delta_{k,Q}(t + O(\text{poly}\log k)) + \text{negl}(k)$. Then* DLPuz *is $\epsilon_{k,Q}(t)$-difficult for all probabilistic algorithms $\mathcal{A}$ running in time at most t.*

PROOF. We prove the theorem using a sequence of games. Let $\mathcal{A}$ be a probabilistic algorithm with running time t. Let S_i be the event that $\mathcal{A}$ wins in game G_i. We will use an adversary $\mathcal{A}$ that wins the puzzle difficulty experiment to construct an algorithm $\mathcal{B}$ that solves the IDL^* problem.

Game G_0.

Let G_0 be the original difficulty game $\mathsf{Exp}^{\mathsf{Diff}}_{\mathcal{A},\mathsf{DLPuz},Q}(k)$. For clarity, we write the full definition of this game:

1. The challenger first runs the Setup algorithm and obtains $s \leftarrow (b, \rho, \phi(n), (x_i, X_i))$ and $params \leftarrow (n, g, g^b)$. s is kept secret by the challenger and the parameters $params$ are supplied to $\mathcal{A}$.
2. Whenever $\mathcal{A}$ issues a $\mathsf{CreatePuzSoln}(N_C)$ query, the challenger first runs the BPV pair generator BPVGen to obtain a pair (a, g^a) and then computes z, v, and an interval $\mathcal{I}$ of length Q in which v lies as in the puzzle description (Figure 1). The challenger returns $(puz, soln) \leftarrow ((z, g^a, g^b, \mathcal{I}), v)$ to $\mathcal{A}$.
3. At any point during the game, $\mathcal{A}$ is allowed to issue a $\mathsf{Test}(N_C^*)$ query for which the challenger, generates a puzzle $puz^* = (z^*, g^{a^*}, g^b, \mathcal{I}^*)$ using $\mathsf{GenPuz}(s, Q, N_{C^*})$ and returns puz^* to $\mathcal{A}$. Then $\mathcal{A}$ may continue to ask $\mathsf{CreatePuzSoln}(N_C)$ queries.
4. Eventually, $\mathcal{A}$ outputs a potential solution $soln^* = v^*$. If $\mathsf{VerSoln}(puz^*, soln^*) = \mathsf{true}$, then the challenger outputs 1, otherwise it outputs 0.

Hence,

$$\Pr\left(\mathsf{Exp}^{\mathsf{Diff}}_{\mathcal{A},\mathsf{DLPuz},Q}(k) = 1\right) = \Pr(\mathsf{S}_0) \ . \tag{8}$$

Game G_1.

In game G_1, we replace the pseudo-random function H_ρ with a truly random function H. This change is indistinguishable due to the pseudo-randomness of H_ρ, so

$$|\Pr(S_0) - \Pr(S_1)| \leq \mathrm{negl}(k) \ . \tag{9}$$

Game G_2.

In game G_2, we insert an IDL^* challenge into the response to the Test query. In particular, the experiment proceeds as follows:

1. Obtain an IDL^* challenge $(g, y, n, \bar{\mathcal{I}})$. Choose a long-term secret $b \leftarrow_R \mathbb{Z}_n$ and compute all other values in Setup as specified in game G_1. Set $params \leftarrow (g, n)$.
2. Run $\mathcal{A}(params)$ with oracle access to $\mathsf{CreatePuzSoln}(\cdot)$ and $\mathsf{Test}(\cdot)$, which are answered as follows:
   - $\mathsf{CreatePuzSoln}(str)$: As in game G_1.
   - $\mathsf{Test}(str^*)$: Use the IDL^* challenge y as g^a. Compute z^* specified, and set $\mathcal{I} \leftarrow \bar{\mathcal{I}} + b \cdot z^*$. Return $puz \leftarrow (N_S, z^*, y, g^b, \mathcal{I})$.
3. $\mathcal{A}$ outputs a potential solution $soln^*$.
4. Output 1 if $g^{soln^*} \equiv y \cdot (g^b)^{z^*} \mod n$ and 0 otherwise.

If $\mathcal{A}$ wins game G_2, then $soln^*$ can be converted into a solution $soln^* - b \cdot z^*$ for the IDL^* challenger. Hence,

$$\Pr(\mathsf{S}_2) \leq \mathsf{Adv}^{\mathsf{IDL}^*}_{\mathcal{B},\mathsf{GenRSA},Q}(k) \tag{10}$$

where $\mathcal{B}$ is our challenger which runs in time $t(B) = t(A) + (N+1)t_{\mathsf{exp}} + c$ where c is a constant.

The messages generated by the challenger in G_2 are identical to those in G_1 except for the following modifications:

- In game G_2, the challenger selects a random $b \in Z_n$, instead of a random $b \in Z_{\phi(n)}$. This change is indistinguishable due to the fact that $(n - \phi(n))/n \approx O\left(1/\sqrt{n}\right)$.
- The value g^a which is returned during the Test query: in G_1 it is an output from the BPV generator BPVGen whereas in G_2 it is uniformly random. By Theorem 1, one can choose N and ℓ so that the distribution of the BPV generator is statistically close to the uniform distribution.

Hence

$$|\Pr(S_1) - \Pr(S_2)| \leq \sqrt{M / \binom{N}{\ell}} + O\left(1/\sqrt{n}\right) \leq \mathrm{negl}(k) \tag{11}$$

for a fixed M, where the second inequality follows from appropriate choices of M, N and ℓ.

Final result.

Combining equations (8) through (11) yields the desired result. □

REMARK 2. *Though our puzzle is proven secure in the Chen* et al. *model, it does not seem straightforward to prove the security of our puzzle in the multiple puzzle difficulty definition of Stebila* et al. *[19]. To prove the security of our puzzle in the Stebila* et al. *model we need to either extend the proposed hardness assumption (to a new variant of the interval discrete logarithm problem for example) or find a suitable computationally hard problem.*

6. PERFORMANCE COMPARISON

The experimental results of the number theoretic puzzles for 512-bit RSA modulus with the security parameter $k = 56$ and the hash based puzzle appear in Table 3. The results are shown for difficulty levels ranging from low, to high. The experiment is run on a single core of a 3.06 GHz Intel Core i3 with 4GB RAM, compiled using `gcc -O2` with architecture x86_64. The big integer arithmetic from OpenSSL 0.9.8*r* is used to implement the software.

It is evident from our experimental results that our puzzle is much faster to verify than the existing number theoretic puzzles. More precisely, for the 512-bit RSA modulus, the solution verification time of DLPuz is approximately 89 times faster when compared with Rivest *et al.* puzzle and by approximately 50 times faster when compared with Karame-Capkun puzzle. In addition, the solution verification time of DLPuz is approximately 3 times faster when compared with Rangasamy *et al.* puzzle. On the other hand, the solution verification time of DLPuz is only 1.4 times slower when compared with Chen *et al.*'s hash based puzzle (which is proven secure in the random oracle model).

Note that the running time of generating DLPuz includes the time to compute HMAC-SHA1 operation. Even though the puzzle generation algorithm GenPuz of our puzzle is 4 to 7 times slower than the GenPuz in Rivest *et al.*, Karame-Capkun and Chen *et al.* puzzles, the cumulative puzzle generation and verification time of our puzzle is still less than the corresponding times in Rivest *et al.* and Karame-Capkun. Moreover, the cost of GenPuz in DLPuz can be further reduced by setting a lower value for ℓ and by increasing the number of precomputed pairs N in the puzzle setup phase.

7. CONCLUSION

Puzzle	Difficulty	Setup (ms)	GenPuz (μs)	FindSoln (s)	VerAuth + VerSoln (μs)
	512-bit modulus, $k = 56$. For DLPuz, $N = 65536$ and $\ell = 8$. For, Rangasamy *et al.* puzzle, $\ell = 4$ and $N = 2500$				
Low Difficulty					
Rivest *et al.* [17]	1 million	13.919	4.80	1.54	474.68
Karame-Capkun [12]	1 million	11.520	8.37	1.59	263.35
Hash based puzzle of Chen *et al.*[6]	2^{22}	0.002	5.92	1.07	3.77
Rangasamy *et al.*[16]	1 million	1401.14	16.66	1.54	14.75
DLPuz	10 million	31863	31.437	1.05	5.31
Medium Difficulty					
Rivest *et al.* [17]	10 million	49.989	4.80	15.17	474.83
Karame-Capkun [12]	10 million	28.951	8.37	15.18	265.28
Hash based puzzle of Chen *et al.*[6]	2^{26}	0.002	5.92	16.84	3.77
Rangasamy *et al.*[16]	10 million	1419.78	16.66	15.34	14.53
DLPuz	150 million	31832	32.01	18.10	5.29
High Difficulty					
Rivest *et al.*[17]	100 million	416.292	4.81	157.10	470.61
Karame-Capkun [12]	100 million	218.757	8.35	160.97	259.39
Hash based puzzle of Chen *et al.*[6]	2^{29}	0.002	5.87	134.38	3.77
Rangasamy *et al.*[16]	100 million	1609.83	16.76	158.22	14.88
DLPuz	1500 million	31885	32.01	175.41	5.27

Table 3: Timings for number theoretic puzzles and hash based puzzles.

Client puzzles are a promising countermeasure for defense against denial of service attacks. Hash-based puzzles are very efficient but are generally secure only in the random oracle model. On the other hand, number-theoretic puzzles can be shown secure in the standard model but existing puzzles have had expensive puzzle generation or verification operations. We have presented a number-theoretic client puzzle that is not only efficient but also has a standard model proof of security in the Chen *et al.* model. To prove difficulty of our puzzle, we introduced a new variant of the interval discrete logarithm problem and showed the hardness of this new problem under the factorisation and composite interval discrete logarithm assumptions. Our experimental results show that, for 512-bit modulus, the solution verification time of our proposed puzzle are much faster than the Karame-Čapkun and the Rivest *et al.*'s time-lock puzzle.

Future Work.

Though we show that our puzzle satisfies Chen *et al.*'s security notions, the proof for achieving difficulty in the stronger model of Stebila *et al.* does not follows directly. Hence, constructing an efficient number theoretic standard model puzzle which satisfies the stronger difficulty notion of Stebila *et al.* appears to be an interesting open problem. Additionally, constructing a provably secure hash based puzzle that satisfies the strong definition of Stebila *et al.* in the standard model remains an open problem.

8. REFERENCES

[1] T. Aura and P. Nikander. Stateless connections. In Y. Han, T. Okamoto, and S. Qing, editors, *Proceedings of the First International Conference on Information and Communication Security, ICICS'97, Beijing, China, November 11-14, 1997*, volume 1334 of *Lecture Notes in Computer Science*, pages 87–97. Springer, 1997.

[2] S. Babbage, D. Catalano, C. Cid, O. Dunkelman, C. Gehrmann, L. Granboulan, T. Lange, A. Lenstra, P. Q. Nguyen, C. Paar, J. Pelzl, T. Pornin, B. Preneel, C. Rechberger, V. Rijmen, M. Robshaw, A. Rupp, N. Smart, and M. Ward. ECRYPT yearly report on algorithms and keysizes (2007–2008), July 2008.

[3] A. Back. Hashcash-a denial of service counter-measure. *URL: http://www. hashcash. org/papers/hashcash. pdf*, 2002.

[4] V. Boyko. A pre-computation scheme for speeding up public-key cryptosystems. Master's thesis, Massachusetts Institute of Technology, 1998. Available as `http://hdl.handle.net/1721.1/47493`.

[5] V. Boyko, M. Peinado, and R. Venkatesan. Speeding up discrete log and factoring based schemes via precomputations. In K. Nyberg, editor, *Advances in Cryptology - EUROCRYPT '98,*, volume 1403 of *Lecture Notes in Computer Science*, pages 221–235. Springer, 1998.

[6] L. Chen, P. Morrissey, N. Smart, and B. Warinschi. Security notions and generic constructions for client puzzles. In M. Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009,*, volume 5912 of *Lecture Notes in Computer Science*, pages 505–523. Springer, 2009.

[7] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In E. F. Brickell, editor, *Advances in Cryptology - Proceedings of CRYPTO '92,*, volume 740 of *Lecture Notes in Computer Science*, pages 139–147. Springer, 1992.

[8] S. D. Galbraith, J. M. Pollard, and R. S. Ruprai. Computing discrete logarithms in an interval. Cryptology ePrint Archive, Report 2010/617, 2010. `http://eprint.iacr.org/2010/617`.

[9] R. Gennaro. An improved pseudo-random generator based on discrete log. In M. Bellare, editor, *Advances in Cryptology - CRYPTO 2000,*, volume 1880 of *Lecture Notes in Computer Science*, pages 469–481. Springer, 2000.

[10] M. Jakobsson and A. Juels. Proofs of work and bread pudding protocols. In B. Preneel, editor, *Secure Information Networks: Communications and Multimedia Security, IFIP TC6/TC11 Joint Working Conference on Communications and Multimedia*

Security (CMS '99), September 20-21, 1999, Leuven, Belgium, volume 152 of *IFIP Conference Proceedings*, pages 258–272. Kluwer, 1999.

[11] A. Juels and J. G. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 1999, San Diego, California, USA*, pages 151–165. The Internet Society, 1999. URL: http://www.rsa.com/rsalabs/node.asp?id=2050.

[12] G. Karame and S. Capkun. Low-cost client puzzles based on modular exponentiation. In D. Gritzalis, B. Preneel, and M. Theoharidou, editors, *Computer Security - ESORICS 2010,*, volume 6345 of *Lecture Notes in Computer Science*, pages 679–697. Springer, 2010.

[13] P. Nguyen, I. Shparlinski, and J. Stern. Distribution of modular sums and the security of the server aided exponentiation. In *Proc. Workshop on Cryptography and Computational Number Theory (CCNT'99), Singapore*, pages 257–268. Birkh "auser, 2001.

[14] P. Q. Nguyen and J. Stern. The hardness of the hidden subset sum problem and its cryptographic implications. In M. J. Wiener, editor, *Advances in Cryptology - CRYPTO '99,*, volume 1666 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 1999.

[15] S. Patel and G. S. Sundaram. An efficient discrete log pseudo random generator. In H. Krawczyk, editor, *Advances in Cryptology - CRYPTO '98,*, volume 1462 of *Lecture Notes in Computer Science*, pages 304–317. Springer, 1998.

[16] J. Rangasamy, D. Stebila, L. Kuppusamy, C. Boyd, and J. M. G. Nieto. Efficient modular exponentiation-based puzzles for denial-of-service protection. In *To appear in ICISC 2011 proceedings.*

[17] R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, Cambridge, MA, USA, 1996.

[18] V. Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004. `http://eprint.iacr.org/`.

[19] D. Stebila, L. Kuppusamy, J. Rangasamy, C. Boyd, and J. M. G. Nieto. Stronger difficulty notions for client puzzles and denial-of-service-resistant protocols. In A. Kiayias, editor, *Topics in Cryptology - CT-RSA 2011*, volume 6558 of *Lecture Notes in Computer Science*, pages 284–301. Springer, 2011.

[20] P. C. van Oorschot and M. J. Wiener. On diffie-hellman key agreement with short exponents. In U. M. Maurer, editor, *Advances in Cryptology - EUROCRYPT '96,*, volume 1070 of *Lecture Notes in Computer Science*, pages 332–343. Springer, 1996.

[21] B. Waters, A. Juels, J. A. Halderman, and E. W. Felten. New client puzzle outsourcing techniques for DoS resistance. In V. Atluri, B. Pfitzmann, and P. D. McDaniel, editors, *ACM Conference on Computer and Communications Security*, pages 246–256. ACM, 2004.

APPENDIX

A. EXAMPLE.

As the number of operations required to solve a puzzle ranges from 0 to 2^{25}, it is enough for the server to choose N and ℓ such that the indistinguishability bound is less than or equal to ϵ, the bound for difficulty. Suppose the server wants to fix the difficulty Q of a puzzle to be 2^{20} and the 40-bit security level k with a 512-bit modulus n. Let order of g be M, where M is of length 80-bits. Then, from the above table, it is clear that with 2^{16} (x_i, X_i) pre-computed pairs, the server requires to perform only 7 $(\ell - 1)$ modular multiplications to compute a pair (a, g^a) on-line. Then it can set the interval length to be 2^{40} so that the best known solving algorithm requires $\sqrt{Q} = 2^{20}$ operations and the distinguishability is bounded by 2^{-20}.

B. EXISTING ASSUMPTIONS

We begin by defining the factorisation problem and interval discrete log problem specifically for the RSA composite modulus n.

Given a composite integer n such that n is a product of two k-bit primes p and q, the *factorisation problem* is to compute either p or q. The formal definition is as follows.

DEFINITION 6. *(Factorisation Problem) Let k be a security parameter, let* GenRSA *be modulus generation algorithm, and let $\mathcal{A}$ be a probabilistic algorithm. The experiment is as follows. Run* $\mathsf{GenRSA}(1^k)$ *to obtain (n, p, q), and then run $\mathcal{A}$ on input n. The adversary wins the experiment if it outputs either p or q (one of the non-trivial factors of n). We define the advantage of $\mathcal{A}$ in violating the factorisation assumption as*

$$\mathsf{Adv}^{\mathsf{Fact}}_{\mathcal{A},\mathsf{GenRSA}}(k) = \Pr\left(x = p \text{ or } q : (n,p,q) \leftarrow \mathsf{GenRSA}(1^k), x \leftarrow \mathcal{A}(n)\right)$$

Recent recommendations on RSA key sizes [2] indicate that the time required to factor an m-bit RSA modulus is $2^{s(m)}$, where

$$s(m) = \left(\frac{64}{9}\right)^{\frac{1}{3}} \log_2(e)(m \ln 2)^{\frac{1}{3}} (\ln(m \ln 2))^{\frac{2}{3}} - 14 \ .$$

The *composite interval discrete logarithm problem* IDL is to compute x given an RSA modulus $n = pq$, an element $y = g^x \mod n$ for a random x where g is a random element in $\mathbb{Z}_n^*$ and an interval $\mathcal{I}$ of length q such that $x \in \mathcal{I}$. Formally:

DEFINITION 7. *(Composite Interval Discrete Logarithm Problem* IDL*) Let k be a security parameter, q be a difficulty parameter, and* GenRSA *be a modulus generation algorithm. Let $\mathcal{A}$ be a probabilistic algorithm. Define the experiment* $\mathsf{Exp}^{\mathsf{IDL}}_{\mathcal{A},\mathsf{GenRSA},q}(k)$ *as follows:*

1. $n \leftarrow \mathsf{GenRSA}(1^k)$.
2. $g \leftarrow_R \mathbb{Z}_n^*$, $x \leftarrow_R [1, \phi(n)]$, $y \leftarrow g^x \mod n$.
3. $r \leftarrow_R [0, q-1]$, $\mathcal{I} \leftarrow [x - r, x - r + q]$.
4. $x' \leftarrow \mathcal{A}(g, y, n, \mathcal{I})$.
5. *Output* 1 *if* $x' = x$ *and* 0 *otherwise.*

The advantage of $\mathcal{A}$ in violating the IDL *assumption is*

$$\mathsf{Adv}^{\mathsf{IDL}}_{\mathcal{A},\mathsf{GenRSA},q}(k) = \Pr\left(\mathsf{Exp}^{\mathsf{IDL}}_{\mathcal{A},\mathsf{GenRSA},q}(k) = 1\right) \ .$$

Galbraith *et al.* [8] have given the best algorithms to date for solving the Interval Discrete Logarithm problem in a group-agnostic manner, which have an average case expected running time of $(1.660 + o(1))\sqrt{q}$. In groups over a composite modulus n where factoring n is hard, this remains the expected running time.

C. UNFORGEABILITY OF DLPuz

DEFINITION 8. *(Puzzle Unforgeability [6]) Let k be a security parameter, $\mathcal{A}$ be a probabilistic algorithm, and* Puz *be a client puzzle. Define the experiment* $\mathsf{Exp}^{\mathsf{UF}}_{\mathcal{A},\mathsf{Puz}}(k)$ *as follows:*

1. $(params, s) \leftarrow \mathsf{Setup}(1^k)$.
2. *Run* $\mathcal{A}(params)$ *with oracle access to* $\mathsf{CreatePuz}(\cdot)$ *and* $\mathsf{CheckPuz}(\cdot)$*, which are answered as follows:*
   - $\mathsf{CreatePuz}(str, Q)$*: $puz \leftarrow \mathsf{GenPuz}(s, Q, str)$. Return puz to $\mathcal{A}$.*
   - $\mathsf{CheckPuz}(puz)$*: If puz was not an output for any of the* $\mathsf{CreatePuz}(str)$ *query made previously and* $\mathsf{VerAuth}(s, puz) = \mathsf{true}$ *then stop the experiment and output* 1*. Otherwise, return* false *to $\mathcal{A}$.*
3. *Output* 0.

We say that $\mathcal{A}$ wins the game if $\mathsf{Exp}^{\mathsf{UF}}_{\mathcal{A},\mathsf{Puz}}(k) = 1$ *and loses otherwise. The advantage of $\mathcal{A}$ is defined as:*

$$\mathsf{Adv}^{\mathsf{UF}}_{\mathcal{A},\mathsf{Puz}}(k) = \Pr\left(\mathsf{Exp}^{\mathsf{UF}}_{\mathcal{A},\mathsf{Puz}}(k) = 1\right) \ .$$

A puzzle Puz *is said to be* unforgeable *if this advantage is negligible in k for all probabilistic algorithms $\mathcal{A}$ running in time polynomial in k.*

In this unforgeability experiment, the adversary is allowed to query the CreatePuz oracle by choosing a *str* and a puzzle difficulty level Q at will. This is to ensure that even after seeing puzzles with different difficulty levels, the adversary cannot create a valid looking puzzle.

THEOREM 4 (UNFORGEABILITY OF DLPuz). *The client puzzle* DLPuz *is unforgeable.*

PROOF. We prove the theorem using a sequence of games. Let $\mathcal{A}$ be a probabilistic algorithm with running time t. Let S_i be the event that $\mathcal{A}$ wins in game G_i.

Game G_0.

Let G_0 be the original unforgeability game $\mathsf{Exp}^{\mathsf{UF}}_{\mathcal{A},\mathsf{DLPuz}}(k)$. Then

$$\Pr\left(\mathsf{Exp}^{\mathsf{UF}}_{\mathcal{A},\mathsf{DLPuz}}(k') = 1\right) = \Pr(\mathsf{S}_0) \ . \tag{12}$$

Game G_1.

In this game, we modify game G_0 by replacing the HMAC H_ρ with a truly random function H to compute z. This change has a negligible effect on adversary $\mathcal{A}$ because of the pseudo-randomness of HMAC H_ρ. Hence,

$$|\Pr(S_0) - \Pr(S_1)| \leq \mathsf{Adv}^{HMAC}_{\mathcal{B}}(k) \leq \mathrm{negl}(k) \tag{13}$$

where $\mathcal{B}$ is an algorithm running in time $O(t)$, and the second inequality follows whenever H_ρ is a pseudo-random function.

Since the function H in game G_1 is truly random, the probability that an adversary without access to H can guess an output is negligible:

$$\Pr(S_1) \leq \frac{1}{2^k} \ . \tag{14}$$

Combining equations (12)–(14), we obtain the final result, that the adversary's success in forging a puzzle is negligible. □