Effort-Release Public-Key Encryption from Cryptographic Puzzles

Jothi Rangasamy, Douglas Stebila, Colin Boyd, Juan Gonzalez Nieto, and Lakshmi Kuppusamy

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

Abstract. Timed-release cryptography addresses the problem of "sending messages into the future": a message is encrypted so that it can only be decrypted after a certain amount of time, either (a) with the help of a trusted third party time server, or (b) after a party performs the required number of sequential operations. We generalise the latter case to what we call *effort-release public key encryption (ER-PKE)*, where only the party holding the private key corresponding to the public key can decrypt, and only after performing a certain amount of computation which may or may not be parallelisable. Effort-release PKE generalises both the sequential-operation-based timed-release encryption of Rivest, Shamir, and Wagner, and also the encapsulated key escrow techniques of Bellare and Goldwasser. We give a generic construction for ER-PKE based on the use of moderately hard computational problems called puzzles. Our approach extends the KEM/DEM framework for public key encryption by introducing a difficulty notion for KEMs which results in effort-release PKE. When the puzzle used in our generic construction is non-parallelisable, we recover timed-release cryptography, with the addition that only the designated receiver (in the PKE setting) can decrypt.
Keywords: puzzles, difficulty, timed-release encryption, key escrow

1 Introduction

Until 1992, only the *hard* problems of computational complexity were considered as the foundation of cryptography. Dwork and Naor introduced the notion of *moderately hard* problems in 1992 [9]. Since then, moderately hard problems have shown a great deal of promise and have emerged as an important pillar of cryptography. A moderately hard problem is defined as a cryptographic problem such that solving it is not computationally infeasible but also not easy. Moderately hard problems have found their main application in guarding against resource exhaustion attacks such as denial-of-service (DoS) and spam [9, 10]. In these applications, they are called *client puzzles* or *proofs of work*; in the case of DoS attacks, a defending server can force its clients to commit some of its own resources by solving a puzzle, before being granted access to a resource. In this work, we call all such moderately hard problems *cryptographic puzzles* regardless

of who is the solver.

Timed-release cryptography. Rivest *et al.* used a class of cryptographic puzzles to enable "future decryption": encrypting a message so that the decryption is possible only after a certain amount of time has elapsed [14]. They called this idea timed-release cryptography (TRC) and also proposed an alternative way to accomplish TRC with the help of a trusted third party time server. The class of cryptographic puzzles being used for this task is called *time-lock puzzles* and has the following properties: (i) solving them is an intrinsically sequential process (puzzle solving is a non-parallellisable task) and (ii) the puzzle generator may hold a trapdoor allowing an easy (short-cut) way to find solutions. Timed-release encryption has been identified to have many applications in practice. Examples given by Rivest *et al.* include e-voting where opening votes needs to be delayed and sealed-bid auctions where the bids must not be opened until the end of the bidding period.

Motivation. Since Rivest *et al.*'s work, there has been a lot of work on TRC but primarily focused on the second approach of using a trusted third-party server [6, 4, 7]. Moreover the puzzle-based TRC of Rivest *et al.* does not provide confidentiality which is as important as delaying the decryption since anyone can get the message after solving the associated puzzle. Achieving both the confidentiality and the delayed decryption properties is vital in many applications where timed-release encryption is useful. For example, bid-privacy in e-auctions and vote-privacy in e-voting schemes require the addition of confidentiality.

Another interesting scenario is the encapsulated key escrow techniques of Bellare and Goldwasser [1, 2] where both the confidentiality and the delayed decryption properties are essential, but the puzzles need not be non-parallelisable. In particular an Internet service provider may need to escrow (session) keys of its customers to the government law-enforcement agency. To prevent the agency from engaging in massive wire-tapping, puzzles are used to delay the key recovery process. However we observe that puzzle-based TRC has not been treated in a formal way, thus a more formal and thorough approach is desirable.

Contributions: We propose the notion of *effort-release PKE* (ER-PKE) in which *only* the intended recipient can get the message, and that too after a certain amount of computational effort which need not be a sequential process. Our notion generalises both the encapsulated key escrow techniques of Bellare and Goldwasser and the puzzle-based timed-release encryption of Rivest, Shamir, and Wagner in the PKE setting.

Moreover our notion generalises timed-release cryptography in two ways as effort-release cryptography considers not only non-parallellisable puzzles but also parallelisable ones and achieves confidentiality. In particular, the receiver can decrypt the message only after solving the puzzle correctly but the solving process may or may not be parallelisable. Since time-lock puzzles are mainly

non-parallelisable puzzles, restricting effort-release cryptography only to non-parallelisable puzzles recovers timed-release PKE.

In our approach, we adapt the KEM/DEM approach to obtain a generic construction of effort-release PKE. In particular, following this strictly modular approach, we first introduce a difficulty notion for KEMs and quantify the effort required to release the encrypted message by extending the difficulty notion for puzzles by Chen *et al*[5]. We then give a generic construction of difficult KEM as a composition of a PKE and a difficult puzzle. Finally, we define effort-release PKE analogous to difficult KEM, show that difficulty of the KEM carries over to the KEM/DEM hybrid PKE, and provide a concrete construction of ER-PKE.

Paper Outline. Section 2 considers the definition and security notions for puzzles. Section 3 presents the difficulty notion for KEMs by adapting the framework for puzzle schemes. Section 4 is dedicated to effort-release PKE and effort-release hybrid PKE and Section 5 concludes the paper.

Notation. If n is an integer, $|n|$ denotes its length in bits and if S is a set, $|S|$ denotes its cardinality. $a \xleftarrow{\$} S$ means choosing a from the set S at random and if $a = (a_1, \ldots, a_n)$ then $(a_1, \ldots, a_n) \leftarrow a$ means a is parsed as shown. By $y \leftarrow A(x)$, we mean that the output of an algorithm A with the input x is assigned to y; $y \xleftarrow{\$} A(x)$ denotes the similar running of a probabilistic algorithm. PPT means probabilistic polynomial time. $[A(x_1, x_2, \ldots)]$ denotes the set of all possible outputs of A on inputs $x_1, x_2, \ldots$. We use $\mathrm{negl}(\ell)$ to denote an arbitrary function which is negligible as a function of ℓ.

2 Cryptographic Puzzles

The functions that we often use in cryptography are either easy to compute or intractable. In this section we look at a special kind of functions or problems that are moderately hard to compute: cryptographic puzzles.

A cryptographic puzzle scheme CPuz is a tuple $(\mathsf{Setup}, \mathsf{GenPuz}, \mathsf{GetSoln}, \mathsf{FindSoln}, \mathsf{Vrfy})$ of algorithms defined as follows:

$\mathsf{Setup}(1^\ell)$**:** The PPT algorithm that accepts the security parameter ℓ as input and returns output as follows:

- Selects the key space sSpace, the difficulty space QSpace, the string space $\mathsf{strSpace}$, the puzzle instance space $\mathsf{puzSpace}$ and puzzle solution space $\mathsf{solnSpace}$.
- Selects the long-term puzzle secret $s \xleftarrow{\$} \mathsf{sSpace}$.
- Selects the puzzle parameters $\mathsf{params} \leftarrow (\mathsf{sSpace}, \mathsf{puzSpace}, \mathsf{solnSpace}, \mathsf{QSpace})$ required for the client puzzle.
- Returns (params, s)

$\mathsf{GenPuz}(\mathsf{params}, s, Q, str)$**:** Given params, the puzzle secret $s \in \mathsf{sSpace}$, $Q \in \mathsf{QSpace}$ and $str \in \mathsf{strSpace}$ the probabilistic algorithm outputs a puzzle instance $puz \in \mathsf{puzSpace}$.

$\mathsf{GetSoln}(\mathsf{params}, s, puz)$**:** Given params, the puzzle secret $s \in \mathsf{sSpace}$, and a puzzle $puz \in \mathsf{puzSpace}$, the algorithm outputs a solution $soln \in \mathsf{solnSpace}$.

$\mathsf{FindSoln}(\mathsf{params}, puz, \tau)$**:** Given params, a puzzle $puz \in \mathsf{puzSpace}$ and a run time $\tau \in \mathbb{N}$, the algorithm outputs a potential solution $soln \in \mathsf{solnSpace}$ after running for at most τ clock cycles of execution.

$\mathsf{Vrfy}(\mathsf{params}, puz, soln)$**:** is a deterministic algorithm taking as inputs params, a puzzle $puz \in \mathsf{puzSpace}$ and a potential solution $soln \in \mathsf{puzSpace}$ and returns a true or false.

CORRECTNESS. We say a puzzle scheme $\mathsf{CPuz} = (\mathsf{Setup}, \mathsf{GenPuz}, \mathsf{GetSoln}, \mathsf{FindSoln}, \mathsf{Vrfy})$ is correct if for all $(\mathsf{params}, s) \in [\mathsf{Setup}(1^k)]$, all $Q \in \mathsf{QSpace}$, all $str \in \mathsf{strSpace}$ and all $puz \in [\mathsf{GenPuz}(\mathsf{params}, s, Q, str)]$, there exists a $\tau \in \mathbb{N}$ such that $soln \leftarrow \mathsf{FindSoln}(\mathsf{params}, puz, \tau)$, and $\mathsf{true} \leftarrow \mathsf{Vrfy}(\mathsf{params}, puz, soln)$.

Remark 1. The $\mathsf{GetSoln}$ algorithm taking the trapdoor puzzle secret s as an input and will be used by the puzzle generator to find a solution faster than the $\mathsf{FindSoln}$ algorithm. Hence the difficulty of finding solution applies only to the solver running the $\mathsf{FindSoln}$ without the trapdoor s. In this work we are interested in puzzles for which $\mathsf{GetSoln}$ algorithms exist and there exist unique solution for each puzzle instance.

Chen *et al.* [5] were the first to study computational puzzles in a rigorous manner and they introduced two necessary security properties for a puzzle scheme to be effective against DoS attackers. In particular they defined two security notions of puzzles, namely, unforgeability and difficulty. The unforgeability property requires that only the puzzle generator who holds the long-term puzzle secret can generate genuine puzzles. The difficulty property requires that solving a puzzle requires a certain amount of computational work.

In the context of DoS defense the unforgeability property is quite important as argued by Chen *et al.*. In contrast, in the context of ER-PKE a sender generates a puzzle and encrypts it under the receiver's public key so that the adversary does not see puzzles. Hence we require puzzles only to be difficult enough for the intended recipient.

We now describe the puzzle-difficulty game of Chen *et al.* using the code-based game-playing approach due to Bellare and Rogaway [3]. The difficulty of CPuz is defined by the game executed between a challenger and an adversary $\mathcal{A}$ in Figure 1. The advantage of $\mathcal{A}$ playing the difficulty game is defined as

$$\mathbf{Adv}^{Q,\mathsf{Diff}}_{\mathcal{A},\mathsf{CPuz}}(\ell) = \Pr\left[\mathbf{Exec}^{Q,\mathsf{Diff}}_{\mathcal{A},\mathsf{CPuz}}(\ell) = 1\right].$$

Definition 1 (Puzzle-difficulty). *Let $\epsilon_{\ell,Q}(\tau)$ be a family of functions monotonically increasing in τ, where ℓ is a security parameter and Q is a difficulty parameter. Fix $\ell \geq 0$ and $Q \geq 0$. Then, a client puzzle CPuz is $\epsilon_{\ell,Q}(\cdot)$-difficult if*

$$\mathbf{Adv}^{Q,\mathsf{Diff}}_{\mathcal{A},\mathsf{CPuz}}(\ell) \leq \epsilon_{\ell,Q}(\tau),$$

for all $\mathcal{A}$ running in time at most τ ($\tau \in \mathbb{N}$).

$\mathbf{Exec}^{Q,\mathsf{Diff}}_{\mathcal{A},\mathsf{CPuz}}(\ell)$:

1. $(\mathsf{params}, s) \xleftarrow{\$} \mathsf{Setup}(1^\ell)$; $List \leftarrow \emptyset$
2. $(state, str^*) \xleftarrow{\$} \mathcal{A}_1^{\mathcal{O}}(\mathsf{params})$
   - If $\mathcal{A}$ queries $\mathcal{O}(str)$:
     (a) $puz \xleftarrow{\$} \mathsf{GenPuz}(\mathsf{params}, s, str)$
     (b) $soln \leftarrow \mathsf{GetSoln}(\mathsf{params}, s, str, puz)$
     (c) Append $(str, puz, soln)$ to $List$
     (d) Answer $\mathcal{A}$ with $(puz, soln)$.
3. $puz^* \xleftarrow{\$} \mathsf{GenPuz}(\mathsf{params}, s, str^*)$
4. $soln^* \leftarrow \mathsf{GetSoln}(\mathsf{params}, s, str^*, puz^*)$
5. $soln' \xleftarrow{\$} \mathcal{A}_2^{\mathcal{O}}(state, puz^*)$
   - Answer $\mathcal{O}$ queries as above
6. If $(str^*, puz^*, soln')$ is in the $List$, return $\perp$
7. Return 1 if $soln' = soln^*$, else return 0.

Fig. 1. Difficulty experiment for puzzles

Remark 2. Let Puz be an $\epsilon_{\ell,Q}(t)$-difficult puzzle such that each instance of it requires about Q basic steps to solve. Then $\epsilon_{\ell,Q}(t)$ might take the form $t/Q + \mathrm{negl}(\ell)$. However for puzzles we usually have $\ell \geq Q$ and ℓ can be chosen according to the difficulty we aim to achieve. When $\ell \geq Q$, we have that $\epsilon_Q(t) := t/Q + \mathrm{negl}(Q) \geq t/Q + \mathrm{negl}(\ell) = \epsilon_{\ell,Q}(t)$ and therefore an $\epsilon_{\ell,Q}(t)$-difficult puzzle Puz is also $\epsilon_Q(t)$-difficult. Hence, for ease of notation, we set the difficulty parameter Q to be the puzzle security parameter.

In the sections that follow we combine difficult puzzles with public-key primitives to delay the decryption process for a certain amount of time.

3 Difficult Key Encapsulation Mechanism

The basic idea behind the work of Rivest *et al.* on TRC is that the symmetric key used for encrypting the message should not be available immediately for the recipient but only after a certain period of time. Analogous to this approach, the natural extension to a designated solver case is to delay the decryption of the ciphertext encapsulating the symmetric key. This leads us to seek a new class of KEMs and we call a KEM satisfying this goal a *difficult* KEM.

3.1 Definition: Difficult KEM

We now propose the notion of difficult key encapsulation mechanism (DKEM) which will lead to a PKE achieving confidentiality as well as delayed decryption. A DKEM works very similar to a KEM scheme, except that the encapsulation algorithm takes in addition to the regular inputs a secret generated by the parameter generation algorithm. A DKEM is a tuple ($\mathsf{KEM.PG}$,$\mathsf{KEM.KG}$,$\mathsf{KEM.Encap}$, $\mathsf{KEM.Decap}$) of 4 algorithms with the following input/output behavior:

$(\mathsf{params}, s) \xleftarrow{\$} \mathsf{KEM.PG}(1^k, 1^Q)$
$(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KEM.KG}(\mathsf{params})$
$(K, C) \xleftarrow{\$} \mathsf{KEM.Encap}(\mathsf{params}, s, \mathsf{pk})$
$K \leftarrow \mathsf{KEM.Decap}(\mathsf{params}, \mathsf{sk}, C)$.

CORRECTNESS. Correctness of $\mathsf{DKEM} = (\mathsf{KEM.PG}, \mathsf{KEM.KG}, \mathsf{KEM.Encap}, \mathsf{KEM.Decap})$ requires that, for all $(\mathsf{params}, s) \in [\mathsf{KEM.PG}(1^k, 1^Q)]$, and all $(\mathsf{pk}, \mathsf{sk}) \in [\mathsf{KG}(\mathsf{params})]$, we have $\mathsf{KEM.Decap}(\mathsf{params}, \mathsf{sk}, C) = K$ for all $(C, K) \leftarrow \mathsf{KEM.Encap}(\mathsf{params}, s, \mathsf{pk})$ with probability one, where the probability is taken over the coins of $\mathsf{KEM.Encap}$.

DIFFICULTY. We formally define what we mean by a KEM be difficult. We observed that the difficulty in getting session keys is analogous to the difficulty of getting puzzle solutions and thus we extend the puzzle-difficulty property of Chen *et al.* to KEMs as a special security property.

The difficulty of KEM is defined by the game executed between a challenger and an adversary $\mathcal{A}$ in Figure 4. The advantage of $\mathcal{A}$ playing the difficulty game is defined as

$$\mathbf{Adv}^{k,\mathsf{Diff}}_{\mathcal{A},\mathsf{KEM}}(Q) = \Pr\left[\mathbf{Exec}^{k,\mathsf{Diff}}_{\mathcal{A},\mathsf{KEM}}(Q) = 1\right].$$

$\mathbf{Exec}^{k,\mathsf{Diff}}_{\mathcal{A},\mathsf{KEM}}(Q)$:

1. $(\mathsf{params}, s) \xleftarrow{\$} \mathsf{KEM.PG}(1^k, 1^Q)$; $KList \leftarrow \emptyset$
2. $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KEM.KG}(\mathsf{params})$
3. $state \xleftarrow{\$} \mathcal{A}_1^{\mathcal{O}_{\mathsf{Enc}}}(\mathsf{params}, \mathsf{sk})$
   - If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Enc}}$:
     (a) $(C, K) \xleftarrow{\$} \mathsf{KEM.Encap}(\mathsf{params}, s, \mathsf{pk})$
     (b) Append (C, K) to $KList$
     (c) Answer $\mathcal{A}$ with (C, K).
4. $(C^*, K^*) \xleftarrow{\$} \mathsf{KEM.Encap}(\mathsf{params}, s, \mathsf{pk})$
5. $K' \xleftarrow{\$} \mathcal{A}_2^{\mathcal{O}_{\mathsf{Enc}}}(state, C^*)$
   - Answer $\mathcal{O}_{\mathsf{Enc}}$ queries as above
6. If (C^*, K^*) is in the $KList$, return $\perp$
7. Return 1 if $K' = K^*$, else return 0.

Fig. 2. Difficulty experiment for key encapsulation mechanism

Definition 2 (KEM-difficulty). *Let $\epsilon_{k,Q}(\tau)$ be a family of functions monotonically increasing in τ, where k is a security parameter and Q is a difficulty parameter. Fix $k \geq 0$ and $Q \geq 0$. Then, a KEM is $\epsilon_{k,Q}(\cdot)$-difficult if*

$$\mathbf{Adv}^{k,\mathsf{Diff}}_{\mathcal{A},\mathsf{KEM}}(Q) \leq \epsilon_{k,Q}(\tau),$$

for all $\mathcal{A}$ running in time at most τ ($\tau \in \mathbb{N}$).

Remark 3. The difficulty definition of KEM appears quite similar to the non-invertibility under adaptive chosen ciphertext attacks of KEM but differs in the following ways: the encapsulation here takes a secret as an input which is not an input to the decapsulation algorithm, and (hence) the adversary is given access to the encapsulation oracle but not to the decapsulation oracle.

3.2 A Difficult Key Encapsulation Mechanism from Puzzles

As discussed before, cryptographic puzzles have been predominantly used for fighting against resource exhaustion attacks such as junk email (also known as spam) and DoS attacks [9, 10]. Rivest, Shamir and Wagner [14] were the first to combine puzzles with symmetric-key encryption to intentionally delay the message recovery process. In particular, a sender encrypts a message under a (random) symmetric-key. Then the sender generates a puzzle and uses its unique solution to mask the symmetric-key. The ciphertext and the puzzle is sent to the recipient who gets the message after solving the puzzle.

However this technique just delays the decryption process but anyone that is willing to solve the puzzle can acquire the message and hence confidentiality is lost. One natural way to achieve confidentiality is to encrypt the puzzle under the recipient's public key so that anyone else but the recipient cannot see the puzzle. Following this idea, we instantiate a difficult KEM by using a PKE with a difficult puzzle and we then show that the difficulty of the puzzle substantiates the difficulty of the KEM.

The other approach for this idea could be the following: first generate a KEM ciphertext using the recipient's public key, then mask the ciphertext with the solution of a puzzle and send the masked ciphertext and the puzzle to the recipient. Although this approach of adding puzzles separately to the KEM ciphertext looks interesting it may work only for the specific KEM and puzzle schemes. Thus we follow the direct approach of using the puzzle solution to derive a session key and then encrypting the puzzle as it leads to a generic construction of a difficult KEM using a PKE and a difficult puzzle. Moreover, our approach is compatible with practical PKEs such as RSA-REACT as seen in Section 4.2.

The Scheme. Let KDF be a key derivation function [15, 16]. Let (PKE.KG, PKE.Enc, PKE.Dec) be a PKE scheme and let CPuz = (Setup, GenPuz,GetSoln, FindSoln, Vrfy) be a puzzle scheme. Then the proposed DKEM is a tuple (KEM.PG,KEM.KG, KEM.Encap, KEM.Decap) of algorithms as seen in Figure 3:

Security analysis. We consider two security properties for the KEM in Figure 3, namely difficulty (see definition 2) and indistinguishability under adaptive chosen-ciphertext attacks (IND-CCA) [8, 15].

We now show that if CPuz is difficult in the sense of Chen *et al.*, then the DKEM in Figure 3 is difficult according to the definition 2.

$\mathsf{KEM.PG}(1^k, 1^Q)$:

- $(\mathsf{params}, s) \xleftarrow{\$} \mathsf{Setup}(1^Q)$
- Return (params, s)

$\mathsf{KEM.KG}(\mathsf{params})$:

- $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{PKE.KG}(\mathsf{params})$
- Return $(\mathsf{pk}, \mathsf{sk})$

$\mathsf{KEM.Encap}(\mathsf{params}, s, \mathsf{pk})$:

- $puz \xleftarrow{\$} \mathsf{GenPuz}(\mathsf{params}, s, str)$
- $soln \leftarrow \mathsf{GetSoln}(\mathsf{params}, s, puz)$
- $r \xleftarrow{\$} \{0,1\}^{\mathrm{poly}(k)}$; $K \leftarrow \mathsf{KDF}(soln, r)$
- $C \xleftarrow{\$} \mathsf{PKE.Enc}(\mathsf{pk}, puz||r)$
- Return (C, K)

$\mathsf{KEM.Decap}(\mathsf{params}, \mathsf{sk}, C)$:

- $puz||r \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}, C)$
- $soln \leftarrow \mathsf{FindSoln}(\mathsf{params}, puz)$
- $K \leftarrow \mathsf{KDF}(soln, r)$
- Return K

Fig. 3. DKEM from PKE and puzzles

Theorem 1. *Assume that* KDF *is a random oracle. Let* DKEM = (KEM.PG, KEM.KG, KEM.Encap, KEM.Decap) *be the KEM scheme in Figure 3 and let* CPuz = (Setup, GenPuz, GetSoln, FindSoln, Vrfy) *be a difficult cryptographic puzzle scheme according to the definition 1. Suppose there exists an adversary* $\mathcal{A}$ *against the difficulty of* DKEM*, then there is an adversary* $\mathcal{B}$ *against the difficulty of* CPuz *such that*

$$\mathbf{Adv}^{k,\mathsf{Diff}}_{\mathcal{A},\mathsf{DKEM}}(Q) \leq \mathbf{Adv}^{\mathsf{Diff}}_{\mathcal{B},\mathsf{CPuz}}(Q) + \mathrm{negl}(k)$$

and the running time of $\mathcal{B}$ *is asymptotically the same that of* $\mathcal{A}$*.*

Proof. Let $\mathcal{A}$ be the attacker against the difficulty of DKEM that makes at most q_{Enc} queries to $\mathcal{O}_{\mathsf{Enc}}$ and at most q_{KDF} queries to the random oracle KDF. We now build an attacker $\mathcal{B}$ that breaks the difficulty of CPuz using $\mathcal{A}$ and runs in asymptotically the same time as $\mathcal{A}$.

$\mathcal{B}$ interacts individually with the challenger in puzzle-difficulty game and $\mathcal{A}$ playing the KEM-difficulty game. We describe how $\mathcal{B}$ proceeds. $\mathcal{B}$'s input are the public parameters params from the puzzle challenger. Now $\mathcal{B}$ generates $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KEM.KG}(\mathsf{params})$ and invokes $\mathcal{A}$ with $(\mathsf{params}, \mathsf{sk})$.

Encapsulation queries: Now $\mathcal{A}$ may issue a polynomial number of Enc queries for which $\mathcal{B}$ answers as follows: for each of $\mathcal{A}$'s query to $\mathcal{O}_{\mathsf{Enc}}$, $\mathcal{B}$ first selects a string $str \xleftarrow{\$} \mathsf{strSpace}$ and queries $\mathcal{O}$ (the oracle for CreatePuzSoln queries) from the puzzle challenger and in response receives a puzzle-solution pair $(puz, soln)$. Then $\mathcal{B}$ queries the random oracle KDF with $(soln, r)$ for $r \xleftarrow{\$} \{0,1\}^{\mathrm{poly}(k)}$, to get K, computes $C \xleftarrow{\$} \mathsf{PKE.Enc}(\mathsf{pk}, puz||r)$ and responds $\mathcal{A}$ with (C, K). $\mathcal{B}$ records $(soln, r, K)$ into the list it maintains for KDF queries. $\mathcal{B}$ also records $\mathcal{A}$'s queries to the KDF oracle.

Challenge: At some point of time, $\mathcal{A}$ asks for the target ciphertext and now $\mathcal{B}$ selects a random string $str^* \xleftarrow{\$} \mathsf{strSpace}$ and queries the puzzle challenger with

str^* for the target puzzle. In response $\mathcal{B}$ receives the target puzzle puz^* to solve and $\mathcal{B}$ now computes the challenge ciphertext $C^* \xleftarrow{\$} \mathsf{PKE.Enc}(\mathsf{pk}, puz^*||r^*)$ and responds $\mathcal{A}$ with C^*.

At some time, when $\mathcal{A}$ returns a key K' as the output of the game, $\mathcal{B}$ checks with the list if there was an query to KDF having K' as its output. Since $\mathcal{A}$ has non-negligible advantage in breaking the KEM-difficulty it must have queried KDF with some $soln'$ and r^* to obtain K'. In this case $\mathcal{B}$ searches for the tuple matching $(soln', r^*, K')$ and returns $soln'$ to the puzzle challenger playing the difficulty game.

If $\mathcal{A}$ wins the KEM-difficulty game by guessing the session key K' it happens with probability $1/$(the size of key space), which is $\mathrm{negl}(k)$. If $\mathcal{A}$ does not guess the session key but wins the KEM-difficulty game then $\mathcal{B}$ also wins the puzzle-difficulty game. Therefore we have,

$$\mathbf{Adv}^{k,\mathsf{Diff}}_{\mathcal{A},\mathsf{KEM}}(Q) \leq \mathbf{Adv}^{\mathsf{Diff}}_{\mathcal{B},\mathsf{CPuz}}(Q) + \mathrm{negl}(k).$$

The running time of $\mathcal{B}$ is asymptotically the same that of $\mathcal{A}$. □

The following theorem says that if PKE is $\mathsf{IND\text{-}CCA}$-secure, then DKEM in Figure 3 is $\mathsf{IND\text{-}CCA}$-secure. The proof of the theorem is similar to the proof for KEM/DEM by Cramer and Shoup[8] and is omitted due to lack of space.

Theorem 2. *Assume that* KDF *is entropy smoothing key derivation function [15, 16]. Let* $\mathsf{PKE} = (\mathsf{PKE.PG}, \mathsf{PKE.KG}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *be an* $\mathsf{IND\text{-}CCA}$*-secure public-key encryption scheme and let* DKEM *be a KEM as seen in Figure 3. Suppose there exists an* $\mathsf{IND\text{-}CCA}$ *adversary* $\mathcal{A}$ *against* DKEM*, then there exist an adversary* $\mathcal{A}_1$ *against the entropy smoothness of* KDF *and an* $\mathsf{IND\text{-}CCA}$ *adversary* $\mathcal{A}_2$ *against* PKE *such that*

$$\mathbf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{A},\mathsf{DKEM}}(k) \leq \mathbf{Adv}^{\mathsf{ES}}_{\mathcal{A}_1,KDF}(k) + \mathbf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{A}_2,\mathsf{PKE}}(k), \quad \forall k \geq 0$$

where $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *have (asymptotically) the same running time as* $\mathcal{A}$*.*

Remark 4. The proof of Theorem 1 is in the random oracle model where as the proof of Theorem 2 is in the standard model assuming KDF to be entropy smoothing(ES) [15, 16]. It is an interesting open problem to construct a DKEM from a difficult puzzle such that the proof of Theorem 1 is in the standard model.

4 Effort-Release Public Key Encryption

In this section we define difficulty for public-key encryption schemes in analogous to the difficulty for KEMs. We call a PKE scheme having this property an *Effort-Release Public Key Encryption* (ER-PKE). An ER-PKE works similar to a PKE but the recipient holding the decryption key cannot immediately and suddenly complete the decryption process but after the required number of operations. That is, the decryption process requires a certain amount of moderately-hard computation which may or may not be a parallelisable task.

An ER-PKE is a tuple ($\mathsf{PKE.PG}$,$\mathsf{PKE.KG}$, $\mathsf{PKE.Enc}$, $\mathsf{PKE.Dec}$) of 4 algorithms with the following input/output behaviour:

$(\mathsf{params}, s) \xleftarrow{\$} \mathsf{PKE.PG}(1^k, 1^Q)$
$(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{PKE.KG}(\mathsf{params})$
$C \xleftarrow{\$} \mathsf{PKE.Enc}(\mathsf{params}, s, \mathsf{pk}, m)$
$m \leftarrow \mathsf{PKE.Dec}(\mathsf{params}, \mathsf{sk}, C)$.

Correctness. A ER-PKE scheme ($\mathsf{PKE.PG}$,$\mathsf{PKE.KG}$,$\mathsf{PKE.Enc}$, $\mathsf{PKE.Dec}$) is *correct* if, for all $(\mathsf{params}, s) \in [\mathsf{PKE.PG}(1^k)]$, all $(\mathsf{pk}, \mathsf{sk}) \in [\mathsf{PKE.KG}(\mathsf{params})]$, and all plaintexts $m \in \mathsf{MsgSp}$, we have $\mathsf{PKE.Dec}(\mathsf{params}, \mathsf{sk}, \mathsf{Enc}(\mathsf{params}, s, \mathsf{pk}, m)) = m$ with probability one, where the probability is taken over the coins of $\mathsf{PKE.Enc}$.

Informally we call a PKE effort-release if the decryption algorithm cannot be run trivially and the adversary should put in some computational effort for pre-specified expected amount of time for the successful decryption of a ciphertext.

We now formally define what do we mean by effort-release PKE. As for the difficulty of KEM, the effort-release game is defined as the difficulty game executed between a challenger and an adversary $\mathcal{A}$ in Figure 4. The advantage of $\mathcal{A}$ playing the difficulty game is defined as

$$\mathbf{Adv}^{k,\mathsf{Diff}}_{\mathcal{A},\mathsf{PKE}}(Q) = \Pr\left[\mathbf{Exec}^{k,\mathsf{Diff}}_{\mathcal{A},\mathsf{PKE}}(Q) = 1\right].$$

$\mathbf{Exec}^{k,\mathsf{Diff}}_{\mathcal{A},\mathsf{PKE}}(Q)$:

1. $(\mathsf{params}, s) \xleftarrow{\$} \mathsf{PKE.PG}(1^k, 1^Q)$; $List \leftarrow \emptyset$
2. $(\mathsf{pk},\mathsf{sk}) \xleftarrow{\$} \mathsf{PKE.KG}(\mathsf{params})$
3. $state \xleftarrow{\$} \mathcal{A}_1^{\mathcal{O}_{\mathsf{Enc}}}(\mathsf{params}, \mathsf{sk})$
   - If $\mathcal{A}$ queries $\mathcal{O}_{\mathsf{Enc}}(m)$:
     (a) $C \xleftarrow{\$} \mathsf{PKE.Enc}(\mathsf{params}, s, \mathsf{pk}, m)$
     (b) Append (C, m) to $List$
     (c) Answer $\mathcal{A}$ with (C, m).
4. $m^* \xleftarrow{\$} \mathsf{MsgSp}$, the message space
5. $C^* \xleftarrow{\$} \mathsf{PKE.Enc}(\mathsf{params}, s, \mathsf{pk}, m^*)$
6. $m' \xleftarrow{\$} \mathcal{A}_2^{\mathcal{O}_{\mathsf{Enc}}}(state, C^*)$
   - Answer $\mathcal{O}_{\mathsf{Enc}}$ queries as above
7. If (C^*, m^*) is in the $List$, return $\perp$
8. Return 1 if $m' = m^*$, else return 0.

Fig. 4. Difficulty experiment for Effort-Release PKE

Definition 3 (Effort-Release PKE). *Let $\epsilon_{k,Q}(\tau)$ be a family of functions monotonically increasing in τ, where k is a security parameter and Q is a difficulty parameter. Fix $k \geq 0$ and $Q \geq 0$. Then, a PKE is $\epsilon_{k,Q}(\cdot)$-effort-release*

if

$$\mathbf{Adv}^{k,\mathsf{Diff}}_{\mathcal{A},\mathsf{PKE}}(Q) \leq \epsilon_{k,Q}(\tau),$$

for all $\mathcal{A}$ running in time at most τ ($\tau \in \mathbb{N}$).

4.1 Effort-Release Hybrid PKE

Effort-release hybrid PKE can be seen as an extension of Rivest *et al.*'s timed-release (symmetric-key) encryption to the PKE setting. In this section, we give definition for an effort-release hybrid PKE, which works very similar to a hybrid PKE proposed by Cramer and Shoup [8, 15] and we then prove that (i) the hybrid PKE scheme in Figure 5 is an effort-release PKE if the DKEM is difficult according to the definition 2 and (ii) the effort-release hybrid PKE is IND-CCA-secure if both the DKEM and DEM are IND-CCA-secure.

ER-PKE.PG$(1^k, 1^Q)$:

- $(\mathsf{params}, s) \xleftarrow{\$} \mathsf{KEM.PG}(1^k, 1^Q)$
- Return (params, s)

ER-PKE.KG(params) :

- $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KEM.KG}(\mathsf{params})$
- Return $(\mathsf{pk}, \mathsf{sk})$

ER-PKE.Enc$(\mathsf{params}, s, \mathsf{pk}, m)$:

- $(K, c_1) \leftarrow \mathsf{KEM.Encap}(\mathsf{params}, s, \mathsf{pk})$
- $K' \leftarrow \mathsf{KDF}(K)$
- $c_2 \leftarrow \mathsf{DEM.Enc}(K', m)$
- $C \leftarrow (c_1, c_2)$
- Return C

ER-PKE.Dec$(\mathsf{params}, \mathsf{sk}, C)$:

- $(c_1, c_2) \leftarrow C$
- $K \leftarrow \mathsf{KEM.Decap}(\mathsf{params}, \mathsf{sk}, c_1)$
- $K' \leftarrow \mathsf{KDF}(K)$
- $m \leftarrow \mathsf{DEM.Dec}(K', c_2)$
- Return m

Fig. 5. Effort-Release hybrid PKE

Remark 5. In the Effort-Release hybrid PKE in Figure 5, the operation $K' \leftarrow \mathsf{KDF}(K)$ may look redundant and undesirable since K itself is usually the output of KDF. As shown in Theorem 3, having $K' \leftarrow \mathsf{KDF}(K)$ allows us to prove that the difficulty of DKEM implies the difficulty of the hybrid ER-PKE.

Theorem 3. *Assume that* KDF *is a random oracle. Let* DKEM = (KEM.PG, KEM.KG, KEM.Encap, KEM.Decap) *be a difficult KEM scheme according to the definition 2 and let* ER-PKE *be the scheme in Figure 5. Suppose there exists an adversary $\mathcal{A}$ against the difficulty of* ER-PKE*, then there is an adversary $\mathcal{B}$ against the difficulty of* DKEM *such that*

$$\mathbf{Adv}^{k,\mathsf{Diff}}_{\mathcal{A},\mathsf{ER\text{-}PKE}}(Q) \leq (1/q_{\mathsf{KDF}})\mathbf{Adv}^{k,\mathsf{Diff}}_{\mathcal{B},\mathsf{DKEM}}(Q),$$

where q_{KDF} is an upper bound on the number of queries to KDF *made by $\mathcal{A}$ and the running time of $\mathcal{B}$ is asymptotically the same that of $\mathcal{A}$.*

The proof to this theorem is similar to the proof of Theorem 1 and is omitted due to lack of space.

Now the following theorem shows that if both the Difficult-KEM and the DEM are IND-CCA-secure then so is the the Hybrid PKE scheme in Figure 5.

Theorem 4 (Difficult-KEM/DEM Composition Theorem). *Let (*KEM.PG, KEM.KG, KEM.Encap, KEM.Decap*) be an* IND-CCA*-secure* DKEM*, let (*DEM.Enc, DEM.Dec*) be an* IND-CCA*-secure* DEM*, and let (*PKE.PG,PKE.KG,PKE.Enc, PKE.Dec*) be the resulting hybrid ER-PKE scheme. Then, for any* IND-CCA *adversary* $\mathcal{A}$ *against* ER-PKE*, there exists an* IND-CCA *adversary* $\mathcal{B}_1$ *against* DKEM *and an* IND-CCA *adversary* $\mathcal{B}_2$ *against* DEM *such that*

$$\mathbf{Adv}^{\text{IND-CCA}}_{\mathcal{A},\text{ER-PKE}}(k) \leq \mathbf{Adv}^{\text{IND-CCA}}_{\mathcal{B}_1,\text{DKEM}}(k) + \mathbf{Adv}^{\text{IND-CCA}}_{\mathcal{B}_2,\text{DEM}}(k) \quad \forall k \geq 0$$

and $\mathcal{B}_1$ *and* $\mathcal{B}_2$ *have (asymptotically) the same running time as* $\mathcal{A}$*.*

The proof to this theorem is almost identical to the proof for KEM/DEM by Cramer and Shoup [8] and is omitted.

4.2 Constructions of Effort-Release Hybrid PKE

Theorem 3 states that if the underlying KEM is difficult then the resulting hybrid encryption scheme is effort-release PKE. Therefore, as shown in Section 3.2 we can easily instantiate an effort-release hybrid PKE by constructing a difficult KEM from difficult puzzles in [5, 12, 17] and combining it with a DEM.

Constructions of Timed-Release PKE. As shown by Rivest *et al.*[14] timed-release encryption can be obtained from non-parallelisable puzzles in [13, 14]. Therefore using any of these two puzzles in a generic DKEM from Section 3.2 yields a timed-release PKE.

Effort-Release RSA-REACT To get an idea of how a practical effort-release PKE might look, we briefly describe a way of constructing ER-PKE from RSA-REACT proposed by Okamoto and Pointcheval [11]. In particular we instantiate Chen *et al.*'s generic puzzle with SHA1 hash function and combine it with KEM part of RSA-REACT and use AES to implement DEM part of RSA-REACT. The resulting ER-RSA-REACT works as follows:
Let G and H be two hash functions with appropriate domains.

The key generation algorithm $\mathsf{KG}(1^k)$**.** On input of the security parameter k, the probabilistic algorithm generates an RSA modulus n and outputs a public-key (e, n) and a secret-key $(d, \phi(n))$ such that $d = e^{-1} \mod \phi(n)$.

The encryption algorithm $\mathsf{Enc}((e, n), m)$**.** Given a message m and a public key (e, n) the PPT algorithm first picks $r \xleftarrow{\$} \mathbb{Z}_n$ and $u \xleftarrow{\$} \{0,1\}^{\text{poly}(Q)}$. Now it computes $v \leftarrow \mathsf{SHA1}(u)$ and parses u into u_1 and u_2 such that u_2 is of

length Q in bits. Then the algorithm computes $K \leftarrow G(u_2, r)$ and produces a ciphertext (x, c, h) of m, where $x \leftarrow (u_1||v||r)^e \mod n$, $c \leftarrow \mathsf{AES}_K(m)$ and the checking value $h = H(r, m, x, c)$. Output is the ciphertext $C \leftarrow (x, c, h)$.

The decryption algorithm $\mathsf{Dec}((d, \phi(n)), C)$**.** The algorithm first decrypts x using the secret key to obtain $u_1||v||r = x^d \mod n$. Now it starts solving the puzzle (u_1, v) by guessing u_2 such that $v \stackrel{?}{=} \mathsf{SHA1}(u_1||u_2)$. Then recovers the session key $K = G(u_2, r)$ and the plaintext $m = \mathsf{AES}_K(c)$. Finally checks if $h \stackrel{?}{=} H(r, m, x, c)$. If the check fails, outputs $\perp$ to indicate rejection, otherwise outputs the message m.

Remark 6. In ER-RSA-REACT a recipient may securely outsource the puzzle solving process; if this is not desirable in some applications then including the key material r along with u as input to $\mathsf{SHA1}$ for a puzzle generation prevents such outsourcing by the recipient since the (stand-in) puzzle solver will then have enough information to recover the message.

5 Conclusion

Timed-release cryptography has been gaining increased popularity due to its many interesting applications. While already known schemes for this purpose are mainly based on time-servers, the only alternative way appears to be puzzle-based ones where the receiver will be able to decrypt the message after solving a puzzle. To the best of our knowledge, the puzzle-based approach has been treated in an ad hoc fashion. We have proposed the notion of *effort-release PKE* which generalises both the encapsulated key escrow techniques of Bellare and Goldwasser and the puzzle-based timed-release encryption of Rivest, Shamir, and Wagner in the PKE setting. We also gave a generic construction of effort-release PKE by adapting the KEM/DEM approach which is tailored to moderately-hard puzzles and the type of puzzle being used decides whether the obtained ER-PKE is timed-release or not.

However, our generic construction of a difficult KEM has a proof of difficulty in the random oracle model and hence it is an open problem to construct a difficult KEM (from puzzles) having the proof of difficulty in the standard model.

Acknowledgements. The authors are grateful to anonymous referees for their comments. This work is supported by Australia-India Strategic Research Fund project TA020002.

References

[1] M. Bellare and S. Goldwasser. Encapsulated key escrow. Technical Report 688, MIT Laboratory for Computer Science, April 1996. Available at `http://cseweb.ucsd.edu/~mihir/papers/escrow.html`.

[2] M. Bellare and S. Goldwasser. Verifiable partial key escrow. In R. Graveman, P. A. Janson, C. Neumann, and L. Gong, editors, *ACM CCS*, pages 78–91. ACM, 1997.
[3] M. Bellare and P. Rogaway. The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs. In *EUROCRYPT*, pages 409–426, 2006.
[4] K. Chalkias, D. Hristu-Varsakelis, and G. Stephanides. Improved anonymous timed-release encryption. In J. Biskup and J. Lopez, editors, *ESORICS*, volume 4734 of *LNCS*, pages 311–326. Springer, 2007.
[5] L. Chen, P. Morrissey, N. P. Smart, and B. Warinschi. Security notions and generic constructions for client puzzles. In M. Matsui, editor, *Advances in Cryptology – Proc. ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 505–523. Springer, 2009.
[6] J. H. Cheon, N. Hopper, Y. Kim, and I. Osipkov. Provably secure timed-release public key encryption. *ACM Trans. Inf. Syst. Secur.*, 11:4:1–4:44, May 2008.
[7] S. S. M. Chow and S.-M. Yiu. Timed-release encryption revisited. In J. Baek, F. Bao, K. Chen, and X. Lai, editors, *ProvSec*, volume 5324 of *LNCS*, pages 38–51. Springer, 2008.
[8] R. Cramer and V. Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.
[9] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In E. F. Brickell, editor, *Advances in Cryptology – Proc. CRYPTO '92*, volume 740 of *LNCS*, pages 139–147. Springer, 1992.
[10] A. Juels and J. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Proc. Network and Distributed System Security Symposium (NDSS) 1999*, pages 151–165. Internet Society, 1999.
[11] T. Okamoto and D. Pointcheval. REACT: Rapid enhanced-security asymmetric cryptosystem transform. In D. Naccache, editor, *Topics in Cryptology – The Cryptographers' Track at the RSA Conference (CT-RSA) 2001*, volume 2020 of *LNCS*, pages 159–175. Springer, 2001.
[12] J. Rangasamy, D. Stebila, C. Boyd, and J. Gonzalez Nieto. An integrated approach to cryptographic mitigation of denial-of-service attacks. In R. Sandhu and D. S. Wong, editors, *Proc. 6th ACM Symposium on Information, Computer and Communications Security (ASIACCS) 2011*, pages 114–123. ACM, 2011.
[13] J. Rangasamy, D. Stebila, C. Boyd, J. Gonzalez Nieto, and L. Kuppusamy. Efficient modular exponentiation-based puzzles for denial-of-service protection. In *Proc. International Conference on Information Security and Cryptology (ICISC) 2011*, LNCS. Springer, 2011. To Appear. `http://eprints.qut.edu.au/47894/`.
[14] R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. Technical Report TR-684, MIT Laboratory for Computer Science, March 1996. Available on `http://people.csail.mit.edu/rivest/RivestShamirWagner-timelock.pdf`.
[15] V. Shoup. A proposal for an ISO standard for public key encryption (version 2.1). manuscript, 2001. Available on `http://shoup.net/papers`.
[16] V. Shoup. Sequences of games: a tool for taming complexity in security proofs. Technical report, 2004. Available on `http://eprint.iacr.org/2004/332`.
[17] D. Stebila, L. Kuppusamy, J. Rangasamy, C. Boyd, and J. M. González Nieto. Stronger difficulty notions for client puzzles and Denial-of-Service-Resistant protocols. In A. Kiayias, editor, *Topics in Cryptology – The Cryptographers' Track at the RSA Conference (CT-RSA) 2011*, volume 6558 of *LNCS*, pages 284–301. Springer, 2011. `http://eprints.qut.edu.au/40036/`.