GBD Threshold Cryptography with an Application to RSA Key Recovery

February 9, 2005

Abstract

We present protocols for threshold cryptography in the GBD public-key cryptosystem. Both threshold decryption and threshold key generation are covered, in the "honest-but-curious" setting. This shows that it is possible to perform GBD computations in a distributed manner during both key generation and decryption, without revealing the private key to any party. GBD threshold decryption is similar to El-Gamal threshold decryption. GBD threshold key generation is based on adaptations of protocols for RSA key generation by Boneh and Franklin, and Catalano et al, and includes a new protocol for efficiently computing the inverse of a shared secret modulo another shared secret.

We also show an application of GBD threshold cryptography to RSA key recovery. This is based on the use of GBD as a master cryptosystem, whose use allows generation by individual users of RSA moduli that can be factored by using the GBD private key as trapdoor information. This application requires RSA key generation to be tailored, but other operations are standard RSA. Clearly, compromise of the GBD private key would compromise all corresponding RSA private keys, so the security of the GBD master private key should be stronger than the security of the individual RSA keys, and this can be achieved using threshold methods. Finally, we point out two open problems in the RSA key recovery application.

1 Introduction

This paper applies techniques of threshold cryptography to the GBD public-key cryptosystem of González et al [GBD01, GBD04]. We develop both GBD threshold decryption and GBD threshold key generation, and we consider finally an application to key recovery for the RSA cryptosystem.

Threshold cryptography [Gem97] shares the private key amongst a number, ℓ, of players, with a threshold, $t \leq \ell$, such that any subset of $t+1$ or more players can compute (decrypt or sign) with the private key, but no subset of t or fewer players can do so (or indeed deduce any information about either the plaintext or the private key). An example application is the generation by an organisation of a digital signature for a contract, with the organisation's signing key being held not by one trusted individual, but as a number of shares held by members of management, at least $t+1$ of whom need to take part in signing the contract. It has been widely studied since [DF90] developed the first practical scheme.

These schemes offer protection against passive adversaries: the corruption of up to t players does not compromise the security of the private key or allow an adversary to operate with the private key, provided that the adversary is limited to viewing the state and communication messages of the corrupted players. A subsequent development was the addition of *robustness*, providing protection also against active adversaries that can alter the behaviour of corrupted players in arbitrary ways. This is provided by requiring a player to prove that it has carried out its computation correctly without compromising the confidentiality of its share, using a

Verifiable Secret Sharing (VSS) scheme. See, for example, [Ped91] for ElGamal and [Sho00] for RSA.

As described so far, threshold cryptography requires a *trusted dealer* to generate the keys, compute the shares, and communicate these securely to all players. This is a disadvantage in critical applications, leading to the development of schemes in which the players jointly generate the shares of the private key without that private key ever being revealed to any party. This was achieved by [Ped91] for ElGamal, but was more difficult for RSA, with the first practical solution being produced by Boneh and Franklin [BF01], and a robust version by [FMY98].

In this paper, we apply the methods of threshold cryptography to the GBD cryptosystem. The motivation for this work is two-fold. Firstly, we wish to show that applications using GBD can benefit from threshold methods. Secondly, we demonstrate how, by using GBD as a "master" cryptosystem whose security is enhanced by threshold cryptography, it is possible for individuals in an organisation to generate RSA public and private keys that are then used in standard fashion, but with the added capability of allowing the RSA private key to be recovered by a sufficiently large subset of a group of designated key recovery entities.

The GBD threshold methods we develop in this paper are based in part on previous work for RSA and ElGamal. GBD threshold decryption is similar to ElGamal threshold decryption. GBD key generation is based partly on the work of [BF01] to generate a public value which is the product of two secret primes, and partly on ideas from a protocol of Catalano et al [CGH00].

The ability to use GBD to provide RSA key recovery is based on the idea of "self-escrowed" systems introduced by [PY99], in which the private key can be recovered by an authority using the public key plus some trapdoor information available only to the authority. In the case of GBD, the trapdoor information is the GBD master system's private key, possession of which allows factorisation of the RSA modulus $n = pq$ provided that the primes p and q have been chosen from subgroups generated by using the GBD public key. This application of GBD was described in [GVBD02]. As described there, it depends on a highly-trusted key-recovery authority (KRA) which holds the GBD master key; compromise of the KRA would compromise all RSA private keys under the self-escrowed system. In the present work, we show that the KRA can be implemented as a distributed system using threshold techniques, requiring an adversary to corrupt $t + 1$ out of ℓ players in order to recover the master key. The values of t and ℓ may be chosen to provide the desired degree of security and fault-tolerance. This is analogous to the use of threshold cryptography to generate and control the use of the private key for a Certificate Authority [FY98].

1.1 Communication and Security Model

For both threshold decryption and threshold key generation, we denote the number of players by ℓ and the threshold by t. The protocols for key generation require $t \leq \lfloor (\ell - 1)/2 \rfloor$.

In common with other work in this area, we assume a private communication channel between each pair of players, and an authenticated broadcast channel on which the sender of each message is reliably identified[1].

We restrict ourselves to the case of passive adversaries (often referred to as the "honest-but-curious" model of security), in which players may collude to pool their information in an attempt to discover secrets, but all players follow the protocol. Moreover, we use a static model, in which the adversary chooses the set of players to corrupt before the computation begins.

Robustness could be added, if required, using the general algorithm described for example in [Gol04] to produce a robust protocol from a protocol designed for the passive case; this adds a commitment scheme, zero-knowledge proofs and Verifiable Secret Sharing to the protocol. Alternatively, achieving robustness by using an approach specialised for this application would

[1] These may be implemented using standard cryptographic protocols for privacy and authentication.

no doubt be much more efficient (as was the case for example with [GRJK00]), but this has not been investigated further at this stage.

2 Summary of the GBD cryptosystem

González, Boyd and Dawson [GBD01, GBD04] presented a semantically secure public key cryptosystem which operates in $\mathbb{Z}_P^*$, the multiplicative group of integers modulo a large prime P such that $P = 2N + 1$, where $N = Q_0Q_1$ and Q_0, Q_1 are also prime. The security of the cryptosystem is based on the difficulty of determining whether an element $x \in \mathbb{Z}_P^*$ is a member of the subgroup G_{Q_i} of order Q_i (for i = 0,1) given P and two elements g_0, g_1 of order Q_0, Q_1 respectively. The authors conjecture that the best attack against the GBD scheme is factoring N, hence the primes need to be large enough such that factoring N is hard. In this way GBD is similar to RSA since the key lengths need to be of similar size.

For the following description of the GBD constituent algorithms, we denote G_i as the proper subgroup of $\mathbb{Z}_P^*$ of order i. All operations are assumed to be reduced modulo P unless otherwise instructed.

2.1 Key Generation $\mathcal{G}(1^k)$

This algorithm takes as input a security parameter k, and outputs a public key and corresponding private key. The key generation proceeds as follows:

1. Generate the modulus P such that $P = 2N + 1$, where $N = Q_0Q_1$ and Q_0, Q_1 are each random primes of binary size k.
2. Select elements g_0, g_1 of order Q_0, Q_1 respectively.
3. Compute $\alpha_i \equiv Q_{1-i}(Q_{1-i}^{-1} \bmod Q_i)$.
4. Output the public key $pk = (P, g_0, g_1)$ and the secret key $sk = (\alpha_0, \alpha_1)$.

2.2 Encryption $\mathcal{E}(pk, m)$

This algorithm takes as input a message m an element of the subgroup G_N, and a public key $pk = (P, g_0, g_1)$.

1. Choose two integers r_0, r_1 uniformly at random in $\mathbb{Z}_N$.
2. Compute $v_i = g_i^{r_i}$, an element of G_{Q_i} for $i = 0, 1$.
3. Compute $c_i = mv_{1-i}$ for $i = 0, 1$.
4. Output the ciphertext $c = (c_0, c_1)$.

Figure 1 illustrates the encryption algorithm. In the diagram the coordinate axes represent subgroup components G_{Q_0} and G_{Q_1}. The plane represents G_N. Any element $m \in G_N$ has unique projections m_0 in G_{Q_0} and m_1 in G_{Q_1} such that $m = m_0m_1$. Since $v_i \in G_{Q_i}$, it follows that the projection of c_i in G_{Q_i} is m_i.

It can be shown that the unique projection of any element y of G_N on G_{Q_i} is given by

$$y_i = y^{\alpha_i},$$

which allows us to decrypt an encrypted message, as shown below.

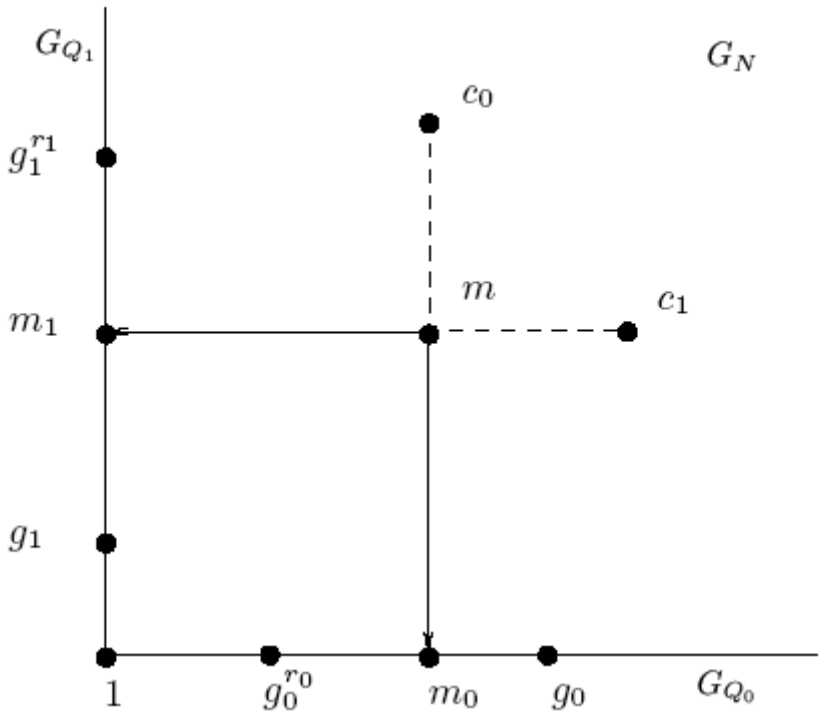

Figure 1: Encryption of $m \in G_n$.

2.3 Decryption $\mathcal{D}(sk, c)$

1. Compute $m_i = c_i^{\alpha i}$ for $i = 0, 1$.
2. Calculate $m = m_0 m_1$.
3. Output m.

It can be seen that encryption takes two modular exponentiations and two modular multiplications, and decryption consists computing the product of two exponentiations. This decryption can actually be implemented using algorithms that are far more efficient than performing the two exponentiations separately. Brown *et al.* [BDG04] examine the implementation of the GBD cryptosystem, and show that the efficiency is comparable to that of a semantically secure ElGamal implementation for equal key lengths.

In fact, the decryption algorithm can be simplified in such a way that it requires only one exponentiation and a division instead of two exponentiations. Let us now describe this modified decryption.

2.4 Decryption $\mathcal{D}'(sk, c)$

1. Compute $u = c_0/c_1$
2. Calculate $m = c_1 * u^{\alpha_0}$.

This decryption is correct since $u = g_1^{r_1} g_0^{-r_0}$ and raising this element of G_N to α_0 projects it onto the subgroup G_{Q_0}, giving $g_0^{-r_0}$. Hence,

$$\begin{aligned} c_1 u^{\alpha_0} &= m g_0^{r_0} . (g_1^{r_1} g_0^{-r_0})^{\alpha_0} \\ &= m g_0^{r_0} . g_0^{-r_0} \\ &= m \end{aligned}$$

Note that only α_0 needs to be used for decryption and hence α_1 does not need to be included in the secret key. If it does need to be used at some later stage, it can be calculated easily since $\alpha_0 + \alpha_1 \equiv 1 \mod N$.

3 Related work in threshold cryptosystems

Here we discuss previous work done in threshold cryptosystems, with particular attention to the work that directly relates to the threshold GBD protocols detailed in later sections. The relevant threshold cryptosystems are ElGamal and RSA, with a focus on threshold decryption and distributed key generation. For the remainder of the paper, we denote a *t-out-of-*ℓ threshold scheme as one where $(t+1)$ is the minimum number of players, from a set of ℓ, required to reveal the shared secret or compute the desired operation.

Desmedt and Frankel [DF90] proposed a non-robust threshold ElGamal decryption scheme based on Shamir's secret sharing [Sha79]. In ElGamal, a ciphertext tuple $C = (g^k, mg^{ak})$ is decrypted by raising g^k to the secret exponent a and finding the inverse to multiply by mg^{ak} to reveal the message m, where g is a generator of a finite field, F_q of prime order q. Hence, a *t-out-of-*ℓ threshold ElGamal system requires the private exponent a to be shared amongst the ℓ players such that any subset T of $(t+1)$, and no less than $(t+1)$, players can collaborate to compute the decryption.

This ElGamal threshold decryption scheme shares the private exponent a amongst the ℓ players by choosing a polynomial $f(x)$ of degree t in a field such that $f(0) = a$. Each player P_s is given the secret share $K_s = f(s)$. Decryption of a ciphertext is done by reconstructing the secret a implicitly in the exponent from $(t+1)$ secret shares using Lagrange interpolation. This means that the secret a is never actually explicitly reconstructed and remains secret. Without loss of generality, let us suppose players 1 to $t+1$ are the subset T wanting to decrypt a message. Each player in T can calculate firstly,

$$a_s = K_s \prod_{r \in T, r \neq s} \frac{(0 - x_r)}{(x_s - x_r)} \tag{1}$$

and then

$$g'_s = (g^k)^{-a_s}.$$

If these g'_s values are then sent to some combiner, or alternatively broadcast to other players, the message can be computed by the combiner, or each player in the latter case by:

$$\begin{aligned} m g^{ak} \prod_{s=1}^{t+1} g'_s &= m g^{ak} g^{k(-a_1 - a_2 - \ldots - a_{t+1})} \\ &= m g^{ak} g^{-ak} \\ &= m. \end{aligned}$$

This scheme assumes there is a trusted dealer to compute and distribute the public key and private key shares. Pedersen [Ped91] built upon this threshold system by adding two major improvements. Firstly, Pedersen provides a means for each participant in the protocol to verify

that his secret share is correct. Secondly, Pedersen describes how to distributively generate the public key and the private key shares, eliminating the need for a trusted party for selecting and distributing the keys. Hence each participant's share is known only by himself and the private key is never reconstituted in one place during either key generation or decryption.

De Santis *et al.* [DSDFY94] presented a RSA threshold scheme for decryption and signatures based, like [DF90], on Lagrange interpolation performed implicitly in the exponent. The problems of interpolating over $\mathbb{Z}_{\phi(n)}$, where n is the RSA modulus and $\phi(n)$ is the secret Euler totient function of n, are avoided by instead working in the field $\mathbb{Z}_{\phi(n)}[X]/\Phi_q[X]$, where $\Phi(n)$ is the qth cyclotomic polynomial (taken mod $\phi(n)$), and q is a prime. Although this allows for standard sharing techniques to be applied directly, the resulting schemes are much more complicated than using a polynomial interpolation over $\mathbb{Z}_{\phi(n)}$. The resulting schemes require either interaction or large share sizes.

Rabin [Rab98] presented a different approach to threshold RSA which aimed to avoid the complexity of the De Santis scheme. His approach combines two pre-existing solutions: (ℓ-*out-of-*ℓ) additive sharing and (t-*out-of-*ℓ) threshold sharing. The private exponent is shared amongst the ℓ players using additive sharing, and each additive share is further shared using a threshold scheme so that if a player is absent, its share can be recovered by $t+1$ players using this back-up scheme.

Shoup [Sho00] presented a RSA threshold signature scheme that is proven to be unforgeable and robust in the random oracle model. Also, Shoup's scheme consists of completely non-interactive signature share generation and verification, and the size of an individual signature share is bounded by a constant times the size of the RSA modulus.

Boneh and Franklin [BF97][BF01] presented protocols for distributive RSA key generation by allowing a number of parties to jointly generate an RSA key. Their contributions include a protocol to generate the modulus $n = pq$ which results in each party P_s holding shares p_s and q_s such that $p = \sum_s^\ell p_s$ and $q = \sum_s^\ell q_s$. They compute n using a simplified version of the BGW protocol [BGW88] and provide a *bi-primality test* for checking whether n is in fact the product of two primes. Algesheimer *et al.* [ACS02] later presented a protocol for generating a random, shared prime, or safe prime, giving rise to a protocol for jointly generating an RSA modulus that is the product of safe primes. Boneh and Franklin also describe an inversion protocol for computing shares of the private exponent $d = e^{-1} \mod \phi(n)$ given the public exponent e. Catalano *et al.* [CGH00] later presented an alternative distributed inversion protocol for this computation that is more efficient, with the drawback of large share sizes.

Frankel, MacKenzie and Yung [FMY98] built upon the Boneh-Franklin solution and described a way of adding robustness to their protocols, and in particular the inversion protocol.

4 GBD Threshold Decryption

The task of distributively decrypting a GBD ciphertext is very similar to that of ElGamal. As we have seen in section 2, GBD decryption requires a division of the ciphertext components and one exponentiation using the private key exponent α_0.

We achieve threshold GBD decryption using a polynomial sharing of the secret α_0. Given a ciphertext $c = (c_0, c_1)$, a subset T of $t+1$ players can collaboratively compute the decryption as follows: each player P_s first computes $u = c_0/c_1$, then computes a_s in a similar way as in (1), where this time the K_s values are $\alpha_{0,s}$, the polynomial shares of α_0, and finally computes $u'_s = u^{a_s}$ and sends this value to each of the other players, or a combining party. Again let us assume that the subset T consists of players 1 to $t+1$. The decryption can then be performed as

$$
\begin{aligned}
c_1 \times \prod_{s \in T}^{t} u'_s &= c_1 . u^{a_1 + a_2 + \ldots + a_{t+1}} \\
&= c_1 . u^{\alpha_0} \\
&= m.
\end{aligned}
$$

Note the decryption protocol as described means that each player needs to know the identities of the other players taking part in the decryption, as its Lagrange coefficient depends on these. An alternative approach would be for each player to compute $u_s = u^{\alpha_{0,s}}$, and for one (or more) players to multiply these to form the decryption after exponentiating each with the appropriate Lagrange coefficient. Such an approach would suit a situation where it is not possible to predict beforehand which set of decryption shares are going to be used — for example if it is necessary to cope with computer and network failures.

Here, we have shown that a polynomial sharing of the private exponent α_0 provides a *t-out-of-*ℓ GBD threshold decryption scheme. In terms of a security analysis of this scheme, since there is a direct relation between our scheme and the Desmedt and Frankel threshold ElGamal decryption scheme, the same security arguments hold for both schemes. To summarise, assuming that the discrete logarithm problem is hard, it can be shown that no extra information about the secret shares a_s can be gained by knowing the u'_s values. Desmedt and Frankel [DF90] use the simulatability concept of zero-knowledge proofs to show this.

4.1 GBD Threshold Projection

It is now useful, for our application to key recovery to be detailed in section 6, to discuss the more general operation of GBD threshold projection. As we have stated in section 2, any element $y \in G_N$ can be uniquely projected onto the subgroup elements $y_0 \in G_{Q_0}$ and $y_1 \in G_{Q_1}$ such that $y = y_0 y_1$. To compute these projections, the following computations should be performed:

$$
\begin{aligned}
y_0 &= y^{\alpha_0} \mod P \\
y_1 &= y / y_0 \mod P
\end{aligned}
$$

To perform this projection when α_0 is shared polynomially as described above, each player can compute y^{a_s} and make this value known to the other players in the subset T. The calculation of y_0 can then be performed simply by computing the product of these y^{a_s} values. The calculation of y_1 can then be performed by each player.

5 GBD Threshold Key Generation

It has long been recognised that the value of threshold decryption is lessened if it depends on a trusted dealer to generate the initial keys and securely distribute the private key shares. A trusted dealer is a single point of vulnerability which knows all the secrets, negating to some degree the value of sharing the private key and using it in a shared fashion. For this reason, there has been a focus on developing schemes which allow the process of generating key shares itself to be carried out in a shared fashion, so that the private key and other secrets of the scheme never exist as complete entities, only as secret shares. This is of particular interest for applications where compromise of the private key would have major consequences, such as generation of the private key for a Certificate Authority [FY98] .

Our motivating example, RSA key recovery based on the use of GBD as a master cryptosystem, is a further example of this. Exposure to an adversary of any one of the GBD secret

parameters Q_0, Q_1 or α_0 would enable the adversary to perform the trapdoor factorisation of all RSA moduli in the system and hence deduce all the RSA private keys. We therefore turn our attention to methods of threshold key generation for GBD.

The GBD threshold key generation described here is based on the work of Boneh and Franklin [BF01] for shared generation of the modulus $N = Q_0Q_1$, plus a new protocol to invert a shared value modulo a shared value, based on Catalano et al [CGH00].

5.1 Building Blocks for Computations on Shared Values

5.1.1 Introduction

In this section we outline the basic protocols on which our shared-value computations are built. In all cases, the input consists of zero, one or two values shared in polynomial fashion, and the output is a single value shared in polynomial fashion. That is, each input and output consists of shares which are values of some polynomial $F(x_s)$ for $s = 1, \ldots, \ell$, where ℓ is the number of players, $F(x)$ is a polynomial of degree t, and $F(0)$ is the shared value. $F(0)$ could of course be determined by Lagrange interpolation from any subset of $t + 1$ shares, but the purpose of the protocols is to carry out the computation without revealing any shared value. A common choice for the x_s is $x_s = s$, however, unless stated otherwise below, the protocols apply to any choice of distinct values for x_s.

An important result by Canetti [Can00] proves that a protocol for computations on shared values can be based on the use of lower-level protocols as building blocks — he calls this *modular composition*. The resulting protocol is secure if the composed operations themselves are secure and the composed operations are used non-concurrently, ie only one is in execution at any one time. Each of the protocols below can be proved secure in Canetti's model (see for example [ACS02]), and the way we compose them in this solution satisfies Canetti's requirement, which is that only a single protocol invocation be in execution at any one time.

5.1.2 Shared Addition

From [BGW88]. This operation requires no interaction; to form the shares of polynomial $C(x) = A(x) + B(x)$, each player sets $C(x_s) = A(x_s) + B(x_s)$.

5.1.3 Sharing a Value

In many protocols, each player needs to share a private value V_s amongst all players. To do so, it generates a polynomial $V_s(x) = \sum_{i=0}^{t} v_{s,i}x^i$, with coefficient $v_{s,i} = V_s$ for $i = 0$ and chosen randomly for $i = 1, \ldots, \ell$. It then sends share $V_s(x_u)$ to player P_u, for $u = 1, \ldots, \ell$.

5.1.4 Revealing a Shared Value

From [BGW88]. To reveal a shared value to one nominated player, at least $t + 1$ players send their shares to that player, who determines the value by Lagrange interpolation. To reveal a shared value to all players, at least $t + 1$ players broadcast their shares, and all players perform the interpolation.

5.1.5 Shared Random Number Generation

From [BB89]. Each player P_s chooses a random value R_s and shares it using the method of 5.1.3. The players then perform a shared addition (5.1.2) to produce the shared value $R = \sum R_s$. Only the final shares $R(x_s)$ are retained.

5.1.6 Share Conversion

Algesheimer et al [ACS02] describe conversions in both directions between additive and polynomial sharings of a value, and also between sharings over the integers and over a finite field. We require a conversion from additive sharing over the integers to polynomial sharing in $\mathbb{Z}_M$, for some M larger than the value A being shared. This is a straightforward combination of methods from [ACS02]. To reshare the value A, held as additive integer shares A_s, requires two steps:

1. Each player P_s computes $A_s \bmod M$ and shares this polynomially using the method of 5.1.3.
2. Each player P_u, having received ℓ values $A_s(x_u)$, computes its polynomial share of A as $A(x_u) = \sum_{s=1}^{\ell} A_s(x_u) \bmod M$.

5.1.7 Shared Multiplication

From [GRR98] and [BBM01]. Given values A and B, shared by polynomials $A(x)$ and $B(x)$, this protocol produces a polynomial $C(x)$ of the same degree, t, which is a sharing of the value $C = AB$. Although the polynomial $A(x)B(x)$ has the required constant term AB, it is of degree $2t$, and moreover its coefficients are not randomly distributed [BGW88]. Gennaro's method [GRR98] solves both problems together:

1. Each player P_s shares the value $C_s = A(x_s)B(x_s)$ using the method of 5.1.3. The result is that each player P_u receives a set of shares $C_s(x_u)$ for $s = 1, \ldots, \ell$.
2. Player P_u then computes its share of $C(x)$ as $C(x_u) = \sum_{s=1}^{\ell} \lambda_s C_s(x_u)$.

Here, the values λ_s are non-zero constants, being the first row of the matrix inverse of the ℓ by ℓ Vandermonde matrix $M = (m_{s,u})$, where $m_{s,u} = x_s^u$. These values clearly depend only on the choice of the x_s. In particular, for the choice $x_s = s$, the values are easily-computed integers [BBM01]:

$$\lambda_s = \frac{(-1)^{s-1}\ell!}{(\ell - s)!s!}$$

5.2 Shared Generation of the GBD Modulus

Boneh and Franklin, in a landmark paper [BF01], present protocols for the shared generation of shared RSA keys, consisting of two parts: firstly the generation of an RSA modulus $n = pq$ where p and q are shared primes, and secondly the generation of the private exponent in shared form. For GBD, we adapt the first part to generate the GBD modulus $N = Q_0Q_1$, adding a test that $P = 2N + 1$ is prime. The second part is specific to RSA; our equivalent for GBD, the generation of the GBD private key, is described in 5.3.

The shared generation of the GBD modulus obtained by adapting [BF01] is summarised in Figure 2. Details of the shared computation steps 1b, 2 and 5 may be found in [BF01]. That paper also contains proofs of the security of each step of the protocol. Since the only change we have made to the protocol is the addition of step 3, which operates on a public value, these proofs apply also to our protocol.

The successful completion of this protocol results in an additive sharing over the integers of Q_0 and Q_1, ie player P_s has shares $Q_{i,s}$ such that $Q_i = \sum_{s=1}^{\ell} Q_{i,s}$. All players also know the value of $N = Q_0Q_1$ and thence $P = 2N + 1$.

The reliance on a biprimality test leads to a larger number of iterations than would be the case for testing each Q_i for primality in isolation. [BF01] quotes an expected number of iterations of 484 to generate a 1024-bit RSA modulus, versus 44 for normal modulus generation

Input: security parameter k.

1. The following steps are performed for Q_0 first and then repeated for Q_1:
   (a) Each player P_s chooses a secret k-bit integer $Q_{i,s}$. The value $\sum_s Q_{i,s}$ is a shared candidate for Q_i.
   (b) The players perform a shared trial division of Q_i by small primes less than some bound B_1. If a factor is found, repeat from step 1a.
2. The players perform a shared computation to reveal the value of $N = Q_0 Q_1$.
3. One of the players performs a primality test on $P = 2N+1$ as a local computation. If this fails, repeat from step 1.
4. One or more players perform further trial divisions on N for small primes in the range $[B_1, B_2]$ for some bound B_2. If a factor is found, repeat from step 1.
5. The players perform a shared computation to test that N is the product of two primes; if this fails, repeat from step 1.

Figure 2: Shared Generation of the GBD Modulus

(both assuming trial divisions up to $B_1 = 8103$ in step 1b). A subsequent implementation paper for RSA [MWB99] describes further optimisations, including a technique of distributed sieving and implementation optimisations based on concurrency. The paper cites total generation times around 1.5 minutes for a 1024-bit modulus on a local network for three players ($\ell = 3$), and 5.6 minutes for $\ell = 5$, using 333 MHz Pentium II systems running Solaris 2.5.1, a 10 MB Ethernet network, and SSL to protect the privacy of communications. These times are completely dominated by modulus generation; the contribution of the subsequent private key generation step is insignificant.

A worst-case estimate for shared GBD modulus generation is obtained by multiplying these times by the expected number of iterations required to find a prime P, about 350 iterations for P of size 1024 bits, potentially giving a GBD modulus generation time of many hours. In practice, it should be considerably smaller, since non-prime P values are eliminated at step 3, before the lengthy biprimality test. Further optimisations are also possible — for example, instead of discarding the two Q_i candidates if step 3 fails, generating one new candidate and testing the primality of P for all pairs of Q_i candidates found so far, like the approach taken by [BDG04]. Even so, GBD threshold key generation using this protocol may be suitable only for applications where it is required infrequently — for example the generation of the keys for the GBD master cryptosystem in the RSA key recovery example.

Note also that [ACS02] presents a shared primality test as a possibly more efficient replacement for the biprimality test of [BF01], but we are not aware of any implementations or experimental results.

5.3 GBD Private Key Calculation

The GBD private key consists of the projection exponent α_0, and is computed from the primes Q_i as $\alpha_0 = Q_1(Q_1^{-1} \bmod Q_0)$. For GBD threshold key generation, the input values Q_i are shared between the players, and we require that the output value α_0 is also shared, to enable GBD

threshold decryption and projection. Shares of α_0 may be kept modulo $N = Q_0Q_1$, since they will be used to exponentiate values $y \in G_N$, which satisfy $y^N \equiv 1 \pmod{P}$. We also require that, in the "honest-but-curious" security model, the shared computation reveals no information to an adversary that it could not obtain from the public key alone.

The secure computation of the private key will be performed as a shared inversion operation followed by a shared multiplication. The multiplication is an application of 5.1.7. We describe the inversion below. Note that the computations require that Q_0 and Q_1 be shared polynomially in $\mathbb{Z}_N$: this is achieved using the method of 5.1.6.

The inspiration for our inversion protocol is that of Catalano et al [CGH00] which was developed for the case where the value to be inverted is known and the modulus shared, and in which computations are done in the integers. We extend the protocol for our case, in which both values are shared, and specialise it for this application by performing computations modulo N. The ability to run this protocol modulo N has attractive consequences: the shares of α_0 have a smaller size, and the security proof is straightforward[2].

Our protocol is based on the shared computation of two random linear combinations of Q_0 and Q_1 modulo N:

$$\begin{aligned} F &= (RQ_0 + SQ_1) \bmod N \\ E &= (TQ_0 + UQ_1) \bmod N \end{aligned}$$

where R, S, T and U are independent shared random values in $\mathbb{Z}_N$. The values of F and E are then revealed to all players, and each player performs a local computation, using the extended GCD algorithm to compute integers a, b and d satisfying $aF + bE = d$. Substituting for F and E, we obtain:

$$\begin{aligned} (aR + bT)Q_0 + (aS + bU)Q_1 &\equiv d \pmod N \\ e(aR + bT)Q_0 + e(aS + bU)Q_1 &\equiv 1 \pmod N \text{ where } e \equiv d^{-1} \pmod N \\ e(aS + bU)Q_1 &\equiv 1 \pmod{Q_0} \text{ since } Q_0 \text{ is a divisor of } N \\ \text{that is, } e(aS + bU) &\equiv Q_1^{-1} \pmod{Q_0} \text{ which is the desired result.} \end{aligned}$$

Each player P_s therefore computes its share of C, the inverse of Q_1, from its shares of S and U as $C_s = e(aS_s + bU_s) \bmod N$. The probability that the inverse $d^{-1} \bmod N$ does not exist is $O(2^{-k})$, which is negligible for realistic values of N. The complete protocol is given in Figure 3.

Theorem 1 *The protocol of Figure 3 securely evaluates the value of* $Q_1^{-1} \bmod Q_0$.

Proof The *correctness* of the evaluation follows immediately from the description above. For *privacy*, we need to show that the protocol reveals no information about Q_0, Q_1 or the result C. Note firstly that the polynomial sharing of Q_0 and Q_1 is obtained using techniques that are known to be secure [ACS02] under the assumptions of Canetti's model. The same applies to the sharing of the intermediate values R, S, T and U. That is, an adversary gains no information about any of these values by obtaining the shares of up to t players.

We now show that publishing F_s and E_s for a specific s reveals no information about the shares $Q_{0,s}$ and $Q_{1,s}$. It can be seen that, for fixed $Q_{0,s}$ and $Q_{1,s}$, and varying R_s and S_s, $F_s = (R_sQ_{0,s} + S_sQ_{1,s}) \bmod N$ ranges over all values in $\mathbb{Z}_N$ except when $Q_{0,s}$, $Q_{1,s}$ and N all have a common factor. Since $N = Q_0Q_1$, this occurs with negligible probability. Moreover, F_s is uniformly distributed over $\mathbb{Z}_N$, being a linear combination in $\mathbb{Z}_N$ of independent random variates which are uniformly distributed over $\mathbb{Z}_N$. This shows that F_s reveals no information about $Q_{0,s}$

[2]An extension of the protocol for the general case where computations must be done in the integers is also feasible, but not required for this application.

1. Polynomially share a random value $R \in \mathbb{Z}_N$, resulting in shares R_s (5.1.5).
2. Similarly share random values S, T, and $U \in \mathbb{Z}_N$, resulting in shares S_s, T_s and U_s.
3. Each player broadcasts the value $F_s = (R_s Q_{0,s} + S_s Q_{1,s}) \bmod N$. These values are shares of a random polynomial $F(x)$ of degree $2t$ which has $F(0) = F = (RQ_0 + SQ_1) \bmod N$.
4. Each player broadcasts the value $E_s = (T_s Q_{0,s} + U_s Q_{1,s}) \bmod N$. These values are shares of a random polynomial $E(x)$ of degree $2t$ which has $E(0) = E = (TQ_0 + UQ_1) \bmod N$.
5. Each player uses Lagrange interpolation (requiring $2t+1$ shares) to compute the values F and E.
6. Each player performs the extended GCD protocol to find a and b such that $aF + bE = d$, and computes $e \equiv d^{-1} \pmod N$.
7. Each player P_s computes its share of $C = Q_1^{-1} \bmod Q_0$ as $C_s = e(aS_s + bU_s) \bmod N$.

An efficient implementation would combine into one round the separate communication rounds implied in steps 1 to 2, and also the rounds in steps 3 to 4. This protocol therefore requires two rounds of communication.

Figure 3: Shared Inversion Protocol

and $Q_{1,s}$. A similar argument applies to E_s. Also, because R_s, S_s, T_s and U_s are independent random variates, F_s and E_s are also independent of one another, and no information is revealed by publishing them together.

In addition we need to show that publishing shares F_s and E_s for all $s \in [1, \ell]$ reveals no information about Q_0 or Q_1. Since the only information that is directly computable from these shares is the values of F and E, we do this by showing that the values of F and E do not reveal any information about Q_0 or Q_1. The argument is similar to that for F_s. The value of $F = (RQ_0 + SQ_1) \bmod N$ ranges over all values in $\mathbb{Z}_N$, except when Q_0, Q_1 and N all have a common factor (meaning that $Q_0 = Q_1$) which has negligible probability. Now we can write

$$F \equiv RQ_0 \pmod{Q_1} \text{ and} \tag{2}$$
$$F \equiv SQ_1 \pmod{Q_0} \tag{3}$$

There are Q_0 different R-values in $\mathbb{Z}_N$ that satisfy equation 2 for a specific value of F, since they are separated by multiples of Q_1. Similarly, Q_1 different S-values satisfy equation 3. So there are $N = Q_0 Q_1$ distinct linear combinations producing a specific value of F. That is, F ranges over all values in $\mathbb{Z}_N$, and there are N pairs (R, S) producing each value F. Since there is a total of N^2 pairs (R, S), all equally probable, we see that each value for F is equally probable, ie F is uniformly distributed over $\mathbb{Z}_N$. By the same argument, so is E. It is clear therefore that publishing F and E reveals no information other than F and E themselves.

Finally, the individual shares C_s themselves which result from the protocol are also uniformly distributed over $\mathbb{Z}_N$, being a linear combination of two uniformly-distributed random values S_s

and U_s. Therefore they satisfy the requirements of polynomial sharing, and any subset of t or fewer shares reveals no information about the inverse C.

5.4 Computing the GBD Public Key

The public key consists of (P, g_0, g_1) where g_0 and g_1 are randomly-chosen elements of the GBD subgroups G_{Q_0} and G_{Q_1}. These may be computed most straightforwardly by choosing a random g in subgroup G_N, projecting g onto G_{Q_0} using $g_0 = g^{\alpha_0} \bmod P$, and computing $g_1 = g \cdot g_0^{-1} \bmod P$. g is most easily obtained by having one player, P_1 say, choose a random h in $[2, P-2]$, computing $g = h^2 \bmod P$ and broadcasting its value. Computing g_0 and g_1 is an application of GBD threshold projection (section 4.1) using the shares $\alpha_{0,s}$ of the private key as computed above.

6 Self-escrowed RSA

The notion of self-escrowed public keys is due to Paillier and Yung [PY99]. A public key encryption scheme $\mathcal{S} = \langle \mathcal{G}, \mathcal{E}, \mathcal{D} \rangle$ is said to be (perfectly) *self-escrowed* when there exists a master encryption scheme $\mathcal{S}' = \langle \mathcal{G}', \mathcal{E}', \mathcal{D}' \rangle$ and a master key pair (e', d') of $\mathcal{S}'$ such that all key pairs of $\mathcal{S}$ satisfy $e = \mathcal{E}'(e', d)$. In other words, a public key is self-escrowed if the public key is itself an encryption of the private key under some master cryptosystem. In the typical application of self-escrowed cryptosystems, a trusted authority known as the key recovery authority (KRA) owns the master key pair. System users employ $\mathcal{S}$ and generate their key pairs according to $\mathcal{G}$. Normally, these keys are then certified by a certification authority (CA). If for whatever reason a private key needs to be restored, e.g. a user loses her private key, the KRA can do so given the corresponding public key. The main advantage of using self-escrowed keys when compared with other proposals (e.g. [YY98, Mic93]) is that the only information that needs to be stored by the KRA is the master key. When the key recovery capability is to be enforced during certification of public keys, it may be required that users prove to the CA that the public keys submitted for certification are indeed self-escrowed, i.e. that they are valid keys from the range of $\mathcal{G}$.

In a self-escrowed public key infrastructure the compromise of the master key is likely to be catastrophic. The secrecy of all users' private keys relies on the secrecy of the master key. Furthermore, users must trust that the KRA will not misuse its key recovery capability. Because of this, it is desirable to distribute the key recovery function among a threshold of authorities.

Paillier and Yung [PY99] described a self-escrowed discrete log based cryptosystem (a variation of ElGamal's cryptosystem over groups of composite order). The master encryption scheme is a deterministic version of Paillier's encryption scheme [Pai99], for which threshold decryption schemes are known [FPS01, DJ01]. In [GVBD02] Gonzalez *et al.* observed that the trap-door function upon which the GBD is built can be used as a trap-door to the prime factorisation of composite numbers of a certain form. This allows the generation of *self-escrowed* factorisation-based public keys (e.g. RSA keys). Now the master encryption is a deterministic version of GBD, for which there was no threshold decryption equivalent until now.

In what follows we apply the threshold GBD key generation and decryption of Sections 4 and 5 to the self-escrowed scheme of González *et al.* [GVBD02] for integer factorisation based cryptosystems. For simplicity we focus on self-escrowed RSA, noting that self-escrowed versions of other factorisation-based cryptosystems can be devised similarly.

6.1 Master cryptosystem

The master cryptosystem is shown in Figure 4 and assumes ℓ KRAs with a threshold of t. The key generation algorithm outputs a description of subgroups G_{Q_0} and G_{Q_1} of $\mathbb{Z}_P^*$ as the public

key, where prime P is of the form $P = 2N + 1$, with $N = Q_0Q_1$, where Q_0, Q_1 are primes of binary length k jointly generated by the KRAs as described in Sect. 5.2. The private key is $\alpha_0 = Q_1(Q_0^{-1} \mod Q_0)$, which is generated and shared in a t-out-of-ℓ fashion among the KRAs as described in Sect. 5.3. Recall that α_0 permits taking the projections of elements of G_N onto G_{Q_0}. $\mathcal{E}'$ is a deterministic version of the GBD encryption algorithm. It is easy to see that its one-wayness is still based on the projection problem assumption. The message space consists of elements of $G_{Q_0} \times G_{Q_1}$. Given a ciphertext c', decryption requires the computation of $c'^{\alpha_0} \mod P$, which can be performed jointly by any t KRAs as shown in Sect. 4.1.

Key Generation $\mathcal{G}'$:

Input: security parameter k

Output: public key $e' = (P, g_0, g_1)$, private-key shares $\alpha_{0,1}, \ldots, \alpha_{0,\ell}$ of α_0.

Encryption $\mathcal{E}'$:

Input: message $(m'_0, m'_1) \in G_{Q_0} \times G_{Q_1}$, public key $e' = (P, g_0, g_1)$.

Output: ciphertext $c' = m'_0 m'_1$.

Decryption $\mathcal{D}'$:

Input: ciphertext c', any subset of $t + 1$ private-key shares from $\alpha_{0,1}, \ldots, \alpha_{0,\ell}$.

Output: message $(m'_0, m'_1) = (c'^{\alpha_0} \mod P, c'/m'_0 \mod P)$.

Figure 4: Master Cryptosystem

6.2 User's cryptosystem

Figure 5 shows the cryptosystem $\mathcal{S}$ employed by users. The only difference with respect to plain RSA [RSA78] is that now the key generation algorithm has been modified to output the prime factors p and q lying in G_{Q_0} and G_{Q_1}. Since for large P the density of prime numbers in the subgroups G_{Q_0}, G_{Q_1} can be considered the same as in the whole group $\mathbb{Z}_P^*$, the prime number theorem (see e.g. [Rie94]) ensures that the amount of primes $(p, q) \in G_{Q_0} \times G_{Q_1}$ is large, and that therefore they can be found efficiently. The RSA moduli generated by $\mathcal{G}$ are ciphertexts of $\mathcal{S}'$ and therefore can be decrypted into its factors by any subset of $t + 1$ KRAs. The key generation algorithm $\mathcal{G}$ can be extended to produce a non-interactive publicly-verifiable zero-knowledge proof that n is the product of two primes belonging to G_{Q_0} and G_{Q_1} as shown in [GVBD02]. Such proof might be required by a CA as a condition of certifying the public key of a user.

6.3 Security of self-escrowed RSA

As noted in [GVBD02], the self-escrowed RSA cryptosystem described above has an important shortcoming that reduces its usefulness. The effective security of RSA moduli n generated by $\mathcal{G}$ is not greater than the security of the integer N generated by $\mathcal{G}'$, while the typical length of n is

Key Generation $\mathcal{G}$:

Input: master public key $e' = (P, g_0, g_1)$

1. Choose random r_0, r_1 in $\mathbb{Z}_{(P-1)/2}$ such that $p = {g_0}^{r_0} \mod P$ and $q = {g_1}^{r_1} \mod P$ are both distinct primes.
2. Compute $n = pq$ in $\mathbb{Z}$.
3. Choose $a \in \mathbb{Z}_{\phi(n)}$.
4. Compute $d = a^{-1} \mod \phi(n)$.

Output: user's public key $e = (a, n)$, user's private key d.

Encryption $\mathcal{E}$:

Input: A message m, public key $e = (a, n)$

Output: ciphertext $c = m^a \mod n$.

Decryption $\mathcal{D}$:

Input: ciphertext c, user's private key d.

Output: plaintext $m = c^d \mod n$.

Figure 5: User's Cryptosystem.

double that of N. Note that the factors p and q of n are chosen randomly from the subgroups G_{Q_0} and G_{Q_1}, and hence their typical length will be approximately $|P|$. It is an open question whether it is possible to modify $\mathcal{G}$ to efficiently generate factors of significantly shorter lengths.

We note that this "key-doubling problem" is a consequence of the fact that the GBD modulus P is known both to RSA clients wishing to generate an RSA key-pair and to the key-generation servers: any participant can attempt on its own to recover the private key of all self-escrowed RSA public keys by factoring P. If P were kept secret, then an attacker would have to fall back on standard factoring algorithms, so the bit size of the RSA modulus size could be reduced to normal levels — e.g. 1024 — and the bit-size of P to half that — e.g. 512.

To achieve this, the following modifications to the above algorithms would be needed, which we leave as future work:

- The algorithm for the generation of the GBD modulus would need to be changed so that N is not revealed to any single server.
- Since P would be a value shared by the servers, the inversion algorithm in Section 5.3 would need to be extended to the case where computations are done over the integers; which appears doable in a way similar to the inversion protocol of Catalano *et al.* [CGH00].
- The key-generation algorithm of the self-escrowed RSA cryptosystem of Figure 5 would need to be distributed, since the factors must be chosen from G_{Q_0} and G_{Q_1}, with the modulus P now being a shared value.

A simpler approach would be to keep P secret only from the RSA clients, not from the key-generation servers, meaning it would be necessary to distribute only the RSA key-generation

algorithm. This would apply in a situation where it was deemed an acceptable risk for P to be known to each server but kept secret otherwise. For example, a client could choose r_i and obtain $g_i^{r_i} \bmod P$ from shares of $g_i^{r_i} \bmod P$ (over the integers) sent by servers. If the client did not trust the servers to compute the right value corresponding to r_i, the servers would have to provide proofs of correctness. A further simplification would be to allow the servers to generate client keys, since clients already trust (a threshold of) servers with the confidentiality of their keys.

Finally, we note that even when the master public key is never exposed, a concern remains as to the security of RSA keys generated by $\mathcal{G}$ due to the sparsity of the distribution of primes from which the factors are chosen. Only a small fraction $O(\frac{1}{\sqrt{P}})$ of primes in $\mathbb{Z}_P$ is eligible to be chosen. It is an open problem to devise a factoring algorithm that takes advantage of the special form of these factors.

7 Conclusion and Further Work

Application to GBD of the techniques of threshold cryptography has been shown to be feasible, thus extending the applicability of GBD to those contexts where it is required to secure the private key information by sharing it between several servers. The threshold key generation protocol may require running times in the order of hours, which would restrict the circumstances in which it is practical. Experimental estimates of the time taken to generate the GBD modulus, and possible improvements in the protocol to reduce this time, are potential areas for future study, as is the development of an efficient robust version of the protocols.

We have demonstrated an application of GBD, and GBD threshold cryptography, to the problem of RSA key recovery by using a self-escrow method. This suffers from the drawback that it requires the RSA key to be double the normal length in order to maintain equivalent security. We are investigating ways to overcome this drawback. In addition there is an open problem of whether a factoring algorithm can be devised to take advantage of the special form of RSA modulus generated by this self-escrow method; clearly this would affect the practical utility of this approach to key recovery.

References

[ACS02] Joy Algesheimer, Jan Camenisch, and Victor Shoup. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In *22nd Annual International Cryptology Conference on Advances in Cryptology, Crypto'02*, volume 2442 of *Lecture Notes in Computer Science*, pages 417–432. Springer-Verlag, 2002. Full version available at `http://eprint.iacr.org/2002/029`.

[BB89] Bar-Ilan, J. and D. Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In *Proceedings of the eighth annual ACM Symposium on Principles of distributed computing*, pages 201–209. ACM Press, 1989.

[BBM01] Carl M. Bender, Dorje C. Brody, and Bernhard K. Meister. Inverse of a Vandermonde matrix, 2001. `http://www.imperial.ac.uk/research/theory/people/brody/DCB/sa6.pdf`.

[BDG04] Jaimee Brown, Ed Dawson, and González Nieto, Juan M. Implementation of the GBD cryptosystem. In *Cryptographic Algorithms and their Uses*, pages 94–109. QUT Publications, 2004.

[BF97] Dan Boneh and Matthew Franklin. Efficient generation of shared RSA keys. In *Advances in Cryptology – Crypto '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 425–445. Springer-Verlag, 1997. Preliminary version of paper published in J. ACM vol 48, 2001.

[BF01] Dan Boneh and Matthew Franklin. Efficient generation of shared RSA keys. *J. ACM*, 48(4):702–722, 2001. A preliminary version was published in Crypto '97, LNCS vol 1294.

[BGW88] Ben-Or, Michael, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault- tolerant distributed computation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 1–10. ACM Press, 1988.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 13(1):143–202, 2000.

[CGH00] Dario Catalano, Rosario Gennaro, and Shai Halev. Computing inverses over a shared secret modulus. In *Advances in Cryptology – Eurocrypt 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 190–206. Springer-Verlag, 2000. Full version with security proofs available from `http://www.research.ibm.com/security/dist-inv.ps`.

[DF90] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In *Advances in Cryptology – Crypto '89*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315. Springer-Verlag, 1990.

[DJ01] Ivan Damgard and Mats Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *PKC '01: Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography*, volume 1992, pages 119–136. Springer-Verlag, 2001.

[DSDFY94] Alfredo De Santis, Yvo Desmedt, Yair Frankel, and Moti Yung. How to share a function securely. In *26th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 522–533, 1994.

[FMY98] Yair Frankel, Philip D. MacKenzie, and Moti Yung. Robust efficient distributed RSA-key generation. In *Thirtieth Annual ACM Symposium on Theory of Computing – STOC '98*, pages 663–672. ACM Press, 1998.

[FPS01] Pierre-Alain Fouque, Guillaume Poupard, and Stern Stern. Sharing decryption in the context of voting or lotteries. In *FC '00: Proceedings of the 4th International Conference on Financial Cryptography*, volume 1962, pages 90–104. Springer-Verlag, 2001.

[FY98] Yair Frankel and Moti Yung. Risk management using threshold RSA cryptosystems. *login:online*, May 1998. `http://www.usenix.org/publications/login/1998-5/frankel.html`.

[GBD01] González Nieto, Juan M., Colin Boyd, and Ed Dawson. A public key cryptosystem based on a subgroup membership problem. In *Third International Conference on Information and Communications Security*, volume 2229 of *Lecture Notes in Computer Science*, pages 352–363. Springer-Verlag, 2001.

[GBD04] González Nieto, Juan M., Colin Boyd, and Ed Dawson. A public key cryptosystem based on a subgroup membership problem. *Designs, Codes and Cryptography*, Accepted for publication 2004.

[Gem97] Peter S. Gemmell. An introduction to threshold cryptography. *CryptoBytes, Technical Newsletter of RSA Laboratories*, 2(3):7–12, 1997.

[Gol04] Oded Goldreich. Foundations of cryptography (a primer), July 2004. `http://www.wisdom.weizmann.ac.il/~oded/foc-sur04.html`.

[GRJK00] Rosario Gennaro, Tal Rabin, Stanislav Jarecki, and Hugo Krawczyk. Robust and efficient sharing of RSA functions. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 13(2):273–300, 2000. Preliminary version published in Crypto '96.

[GRR98] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing*, pages 101–111. ACM Press, 1998.

[GVBD02] González Nieto, Juan M., Kapali Viswanathan, Colin Boyd, and Ed Dawson. A self-escrowed integer factorisation based public key infrastructure. In Santos González and Consuelo Martínez, editors, *VII Spanish Meeting on Cryptology and Information Security*, pages 315–328, Oviedo, Spain, 2002. Universidad de Oviedo.

[Mic93] Silvio Micali. Fair public-key cryptosystems. In Ernest Brickell, editor, *Advances in Cryptology – CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 113–128. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 1993.

[MWB99] Michael Malkin, Thomas Wu, and Dan Boneh. Experimenting with shared generation of RSA keys. In *Proceedings of the Internet Society's 1999 Symposium on Network and Distributed System Security (SNDSS)*, pages 43–56, 1999.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT '99*, number 1599 in Lecture Notes in Computer Science, pages 223–238. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 1999.

[Ped91] Torben Pryds Pedersen. A threshold cryptosystem without a trusted party. In *Advances in Cryptology – Eurocrypt '91*, volume 547 of *Lecture Notes in Computer Science*, pages 522–526. Springer-Verlag, 1991.

[PY99] Pascal Paillier and Moti Yung. Self-escrowed public-key infrastructures. In Jooseok Song, editor, *Information Security and Cryptology–Icisc'99: Second International Conference*, Lecture Notes in Computer Science, pages 249–261, Seoul, Korea, December 1999. Springer-Verlag.

[Rab98] Tal Rabin. A simplified approach to threshold and proactive RSA. In *Advances in Cryptology – Crypto '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 89–104. Springer-Verlag, 1998.

[Rie94] Hans Riesel. *Prime Numbers and Computer Methods for Factorisation*. Birkhäuser, 1994.

[RSA78] Ron Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[Sha79] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, November 1979.

[Sho00] Victor Shoup. Practical threshold signatures. In *Eurocrypt '00*, volume 1807 of *Lecture Notes in Computer Science*, pages 207–218. Springer-Verlag, 2000.

[YY98] Adam Young and Moti Yung. Auto-recoverable and auto-certifiable cryptosystems. In Kaisa Nyberg, editor, *Advances in Cryptology – EUROCRYPT '98*, number 1403 in Lecture Notes in Computer Science, pages 17–31. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 1998.