Distributing the Key Distribution Centre in Sakai–Kasahara Based Systems

Martin Geisler[1] and Nigel P. Smart[2]

[1] Dept. of Computer Science,
University of Aarhus,
IT-parken, Aabogade 34,
DK-8200 Aarhus N,
Denmark
mg@cs.au.dk
[2] Dept. Computer Science,
University of Bristol,
Merchant Venturers Building,
Woodland Road,
Bristol, BS8 1UB,
United Kingdom
nigel@cs.bris.ac.uk

Abstract. One major drawback with using Sakai–Kasahara based identity-based secret keys is that it appears hard to distribute the trust of the key generation centre. For other types of identity based key this distribution can be done using non-interactive and simple techniques. In this short note we explain how this can be done for Sakai–Kasahara style keys using a simple application of a general technique from multi-party computation. The self-checking property of the resulting private key we show can be used to insulate the protocol from malicious adversaries for many practical parameter sizes.

1 Introduction

Distributing the trust placed in trusted third parties is an important problem in the real world, for example one can distribute the signing key for a certificate authority. This is not only an important issue from the point of view of security, but it also helps maintain resilience against network outages or to allow a form of disaster planing. Many of the techniques in this area are based on the method of threshold secret sharing introduced by Shamir in [20]. The techniques used can be applied in many discrete logarithm based situations, and have even been applied to the more complex area of RSA style signature generation [22]. In 1984, Shamir [21] also invented a form of cryptography based on identities as opposed to public keys. There is a similar need for the trusted centre to be distributed in IBE schemes, just as the CA should be in certificate based systems.

Currently, there are three main types of identity based secret key in use, all of which are based on pairings on elliptic curves, namely *full-domain hash*, *commutative blinding* and *exponent inversion*. A classification which was first proposed

M.G. Parker (Ed.): Cryptography and Coding 2009, LNCS 5921, pp. 252–262, 2009.

in [9]. The first set of *full-domain hash* style keys, is typified by the systems of Boneh–Franklin [8] and Sakai et al [17]. In this construction distribution of the key generation centre is easily performed using many of the ideas which have been used in the traditional discrete logarithm based public key setting. The second set, denoted by the term *commutative blinding*, is typified by the Boneh–Boyen encryption scheme [6] (BB-1). Here modifications can be applied to the techniques applied in the full-domain hash case to obtain a distributed key generation centre based on threshold cryptography [7].

The third type of keys are those introduced by Sakai and Kasahara in [18], which are often referred to as those of *exponent inversion* type. These provide the most efficient, in the random oracle model, ID-based encryption scheme known. A basic ID-based scheme was using this concept was proved secure in [11], then a more efficient and conceptually simpler hybrid ID-based encryption scheme was presented in [12]. This key construction can also be used to construct identity based signatures and signcryption [2], and a related construction (also of exponent inversion type) underlies the second Boneh–Boyen encryption scheme (BB-2). Both the schemes in [2] and [12] are in the current IEEE 1363.3 draft.

The problem that this paper investigates is that the exponent inversion type keys currently have no efficient distributed key generation solution. In this paper we present a solution, which requires for each ID-based key a secure multi-party computation to be performed amongst the servers. This is not ideal, for the other types of ID-based keys one does not require a complex distributed key extraction protocol in the distributed generation setting. However, the protocol we require is very simple and requires only one execution of a distributed distributed multiplication protocol modulo q, where q is the size of the underlying groups, followed by each server executing some simple local group computations. We focus on the Sakai–Kasahara type keys, since these are the most relevant for practice (due to the above mentioned standardisation effort), however we will also note the changes which are needed for the BB-2 type keys as well.

The multi-party computation protocol we shall use is in the information theoretic model. The basic ideas in this setting can be traced back to the papers of Ben-Or et al [4] and Chaum et al [10]. The protocol we shall use, using Shamir secret sharing and the evaluation of what is in effect a trivial arithmetic circuit, was presented in [16] and then extended to any linear secret sharing scheme in [13]. The simple technique for deriving modular inverses in this setting, which we use, was first introduced in [1]. The exact protocol details we propose to use can be found explicitly described in the survey by Cramer et al [14]. We present some experimental data on our protocol which makes use of the VIFF system [24].

We end this introduction by pointing out that our protocol is neither deep, nor very high-tech. It is simply applying a known technique from multi-party computation to the problem situation, and then noticing that the resulting protocol is not as inefficient as one usually obtains for multi-party computation protocols for real world problems. What is surprising is that no-one has noticed this before in the community looking at Sakai–Kasahara based pairing protocols.

2 Notation and Problem Statement

Let $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ denote groups of large prime order q, which are equipped with a bilinear pairing,

$$\hat{t}\colon \mathbb{G}_1 \times \mathbb{G}_2 \longrightarrow \mathbb{G}_T.$$

We assume that $\mathbb{G}_1$ and $\mathbb{G}_2$ are generated by P_1 and P_2. We will write $\mathbb{G}_1$ and $\mathbb{G}_2$ additively and $\mathbb{G}_T$ multiplicatively.

We assume there is some global master secret x and we define master public key for the scheme by $R = [x]P_1$. The user secret key extraction algorithm for Sakai–Kasahara keys is to represent an identity ID as an element in $\mathbb{F}_q$ and then compute the public key Q_{ID} and the private key S_{ID} via the equations

$$Q_{\mathrm{ID}} = R + [\mathrm{ID}]P_1,$$
$$S_{\mathrm{ID}} = \left[\frac{1}{x + \mathrm{ID}}\right] P_2.$$

The problem with constructing keys as above is that there is a single point of failure, in that the person who holds the master secret key x is able to compute all secret keys. Hence, any compromise of the computer which holds x leads to a break of the entire system. A standard solution to this problem is to share x amongst a number of computers in such a way that no-one computer can recover x, but that a coalition of the computers can still compute S_{ID}. It is how this calculation is performed that we treat in this paper.

We first define explicitly the parameters of our problem. We let $1 < t \leq m$ denote integers; the value m denotes the number of computers which will share the value of x, whilst t denotes the threshold value, i.e., the maximum number of colluding parties the protocol can tolerate without compromising the privacy. On the other hand $t + 1$ colluding computers will be able to recover the secret x. We will develop a protocol, in the honest-but-curious case, that succeeds as long as more than $2t$ computers are honest-but-curious. For the malicious case we require that more than $3t$ parties take part in the computation. The number of parties which take part in an interaction will be denoted by n, hence $n \leq m$.

We need to define the following three algorithms, each of which take as input the group descriptions $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$.

Setup(): This is a distributed protocol which runs between the m servers and results in each server obtaining a share x_i of the master secret x. The public output of this protocol is the value $R = [x]P_1$. Note, that a coalition of upto t entities should not be able to determine any information about x.

Extract(ID): This is a distributed protocol run between a set of n servers, where $t \leq n \leq m$. It produces n outputs $S_{\mathrm{ID}}^{(i)}$ which are the shares of the secret value S_{ID}. In the case of honest-but-curious parties we require that $n > 2t$, whilst in the case of malicious parties we require that $n > 3t$.

Combine($S_{\mathrm{ID}}^{(1)}, \ldots, S_{\mathrm{ID}}^{(n)}$): This is algorithm is run by the user. It takes the shares produced by the Extract(ID) protocol and produces the valid user secret key S_{ID}.

We assume without loss of generality that the n servers returning a value of $S_{\mathrm{ID}}^{(i)}$, are numbered 1 to n.

The goal in the case of honest-but-curious adversaries is that as long as the number of colluding servers is less than or equal to t, then no information leaks about the underlying master secret x due to any run of the protocol. In the case of malicious adversaries we also wish to ensure that colluding malicious adversaries, not only cannot learn any extra information, they should also not be able to make the user accept an invalid secret key in the combine stage.

One should note that, unlike the Boneh-Franklin or Boneh-Boyen key situation, the Extract(ID) method for Sakai–Kasahara keys which we present is a relatively complicated protocol and not a simple algorithm run independently by each server. We leave it as an open problem to construct a more efficient method to perform the distributed Extract(ID) method.

3 Basic Protocol

In this section we detail exactly how the three protocols above are executed in the case of honest-but-curious adversaries. Our protocol makes use of the t-out-of-n secret sharing scheme of Shamir [20], and the multi-party computation protocol described in [13,14].

Setup(): Using the property that Shamir secret sharing is a PRSS (Pseudo-Random Secret Sharing) scheme the m servers produce a t-out-of-m secret sharing of a random number x modulo q. Underlying this sharing is a polynomial $Q(X)$ modulo q of degree t and each server i obtains the value $x_i = Q(i)$. The servers compute the value $R_i = [x_i]P_1$, and then the master public key is computed by

$$R = \sum_{i=1}^{m} [c_i]R_i$$

where

$$c_i = \prod_{j=1, j\neq i}^{m} \frac{-j}{i-j}.$$

Extract(ID): Assume $n > 2t$ servers are active, they execute the following protocol which applies a technique of [1] to our problem.

- Locally the servers compute a sharing (z_i) of the value of $z = x + \mathrm{ID}$. This can be computed by simply adding the shares (x_i) of x with the public constant ID.
- Again using the PRSS the servers obtain a sharing (r_i) of a random integer r.
- Using a single invocation of the honest-but-curious multiplication protocol from [13,14] the servers compute a sharing (s_i) of $s = z \cdot r \pmod q$.
- The servers recover s by revealing their shares s_i.
- Locally the servers then compute $w_i = r_i/s \pmod q$.
- The servers locally compute $S_{\mathrm{ID}}^{(i)} = [w_i]P_2$, and send this value securely to the requesting user. The servers then terminate.

Note that the values w_i computed by the servers are a sharing of the $w = 1/(x + \mathrm{ID}) \pmod q$. Note, that we cannot reveal w_i directly since that would allow the parties to recover x, hence w is revealed indirectly via the values of $S_{\mathrm{ID}}^{(i)}$. This does not cause a problem due to our Combine method.

$\mathsf{Combine}(S_{\mathrm{ID}}^{(1)}, \ldots, S_{\mathrm{ID}}^{(n)})$: The user secret key is then recovered via

$$S_{\mathrm{ID}} = \sum_{i=1}^{n} [c_i'] S_{\mathrm{ID}}^{(i)}$$

where

$$c_i' = \prod_{j=1, j \neq i}^{n} \frac{-j}{i-j}.$$

The above protocol assumes, due to the multiplication protocol used in the $\mathsf{Extract}(\mathrm{ID})$ stage, that $2t + 1 \leq n$. So for the minimal value of $t = 1$, we have that n must be at least three. From general results on multi-party computation such a bound is the best possible [4,10].

3.1 Security Analysis

We argue that the above protocol provides a secure distributed key generation process in the presence of honest-but-curious adversaries. Firstly, note that the distribution of keys (both master keys and user secret keys) is identical to the situation where we have a single authority. Secondly, note that the underlying multi-party computation protocol is information theoretically secure against honest-but-curious adversaries (assuming at most t dishonest servers).

Each user in the combine stage obtains their secret key, yet the shares they obtain of this key reveal no other information, due to the properties of the underlying secret sharing scheme. To see this we note there is a trival reduction from a the view of the user in the distributed key generation situation to one in the non-distributed key generation situation: In the distributed game the simulator simply extracts the private key S_{ID} from the non-distributed game, and then generates at random a Shamir secret sharing (s_i) of zero. The random shares in the distributed game are then given by

$$S_{\mathrm{ID}}^{(i)} = [s_i] P_2 + S_{\mathrm{ID}}.$$

Note that the discrete logarithms of the $S_{\mathrm{ID}}^{(i)}$ form a random Shamir secret sharing of the discrete logarithm of S_{ID}, thus the simulation is perfect.

4 Security against Malicious Adversaries

In moving to the case of malicious adversaries one needs to deal with two issues. Firstly the servers could behave maliciously in the multiplication protocol step. Secondly, they may carry out this step correctly, yet attempt to subvert the key

generation protocol by passing an incorrect share back to the user. We deal with these two issues in two distinct ways.

The first issue simply requires that we replace the honest-but-curious multiplication protocol with one which is secure against malicious adversaries. This requires that we must have $n > 3t$, and the number of honest participants is equal to at most $n-t$, (since $t+1$ dishonest participants can trivially break the protocol). This ensures that at least $n-t$ parties at the end of the protocol run obtain a valid sharing of the w. All parties then produce $S_{\mathrm{ID}}^{(i)} = [w_i]P_2$ as before. Except, we are only guaranteed that $n-t$ of these values are correct, since up to only at least $n-t$ parties are honest and hence will validly follow the protocol, this is the our second issue.

Solving this seems to pose a major stumbling block, however, we side-step this issue by using the fact that Sakai–Kasahara keys are self checking. Namely, for a valid pair of public/private keys $(Q_{\mathrm{ID}}, S_{\mathrm{ID}})$ we have that

$$\hat{t}(Q_{\mathrm{ID}}, S_{\mathrm{ID}}) = \hat{t}(P_1, P_2).$$

We also use that fact that *in practice* the values of t and n will not be that large, typical examples could indeed be $t = 1$ and $n = 4$. We thus present a different Combine algorithm for our protocol which runs in time $O({}^nC_{n-t})$. This is clearly inefficient for large values of n, yet for the values one could consider in a practical application it is very manageable.

Hence, we can execute the following version of the Combine algorithm. The combiner goes through each of the ${}^nC_{n-t}$ subsets of $\{S_{\mathrm{ID}}^{(1)}, \ldots, S_{\mathrm{ID}}^{(l)}\}$ of size $n-t$, and performs the recombination step with this subset only. The recombined key is then verified to be correct. If it is correct then we terminate, otherwise the next subset of size $n-t$ is taken. As long as at least $n-t$ of the servers are honest the above protocol will output the correct secret key.

If we examine the situation where we keep n to be as small as possible, i.e. $n = 3t+1$, then the workload of this step is equal to

$$W_t = \frac{(3t+1)!}{(2t+1)! \cdot t!}$$

applications of the combining algorithm. For small values of t we find that this work-effort is given by the values in the following table.

t	1	2	3	4	5
W_t	4	21	120	715	4368

We note that this operation is not performed by the servers, but is performed by the client on receipt of the shares from the servers. Hence, the increased time may not be considered too much of a burden as it is only performed once per secret key request, and is performed by the requestor. Additionally, the requestor can trade space for time when recombining different $S_{\mathrm{ID}}^{(i)}$ subsets. Firstly, the $[c_i']S_{\mathrm{ID}}^{(i)}$ values can computed only once and reused, and secondly, having computed S_{ID} from the subset $\{S_{\mathrm{ID}}^{(1)}, \ldots, S_{\mathrm{ID}}^{(l)}\}$ it is easy to compute S_{ID} from $\{S_{\mathrm{ID}}^{(2)}, \ldots, S_{\mathrm{ID}}^{(l+1)}\}$

by subtracting $[c'_1]S_{\mathrm{ID}}^{(1)}$ and adding $[c'_{l+1}]S_{\mathrm{ID}}^{(l+1)}$. All subsets can be computed in this fashion using only a field subtraction and addition for each subset.

We note that whilst this might not be a polynomial time solution, it is very efficient for the type of values which would be used in practice. Also it is protecting against a denial-of-service attack by the malicious servers against the clients, as opposed to the malicious servers trying to recover the underlying master secret key. We also note that most key centre distribution techniques for certificate based schemes are only secure in the honest-but-curious model, since key distribution is used more for resilience and the servers are themselves heavily protected.

5 Implementation Results

The question arises as to whether the above protocol will be suitable in a practical situation. A key generation centre for an identity based cryptographic system will need to produce a large number of keys for distinct identities ID. Hence, if the protocol is too slow then the protocol will not be suitable in real life situation.

Recently a number of practical systems for performing secure multi-party computation have been developed. Of specific interest in our situation if the VIFF (Virtual Ideal Functionality Framework) system [24,15], which itself grew out of the SIMAP [23] and SCET [19] systems. The SIMAP system has been used very successfully in practice, most notably for the Danish sugar beat auction [5].

VIFF is a fully asynchronous framework for specifying secure multi-party computations. It is implemented as a Python library. Each player executes a Python program, and the programs communicate using standard SSL connections. VIFF provides the communication infrastructure and computations primitives.

The implementation of the basic protocol from Section 3 is shown below as a function called extract. The function references a global variable runtime, which is an instance of the central Runtime class provided by VIFF. Among other things, this class provides methods for generating Shamir shares using PRSS (prss_share_random) and for reconstructing such shares (open). Due to extensive use of operator overloading, the addition (x_i + ID) and the multiplication (z_i * r_i) are in fact done on secret shared values, the latter invoking a secure multiplication protocol.

```
def extract(ID, x_i):
    z_i = x_i + ID
    r_i = runtime.prss_share_random(Zq)
    s_i = z_i * r_i
    s = runtime.open(s_i)
    w_i = gather_shares([r_i, s])
    w_i.addCallback(lambda (r_i, s): r_i / s)
    return w_i
```

After opening s_i the players wait until until r_i and s are ready. The result w_i is prepared and an anonymous function is added to a list of callbacks; this function is executed to make the local division.

The Zq variable represents a finite field with a 256 bit prime modulus. The modulus used is

$$q = \texttt{0x80AE401A5230143EFC6ADD72979CAEADB078F3F2EFD3EE07C901B94BC45C61F5}.$$

This is the group order of a Barreto-Naehrig curve [3], and is typical of a group order which would be used in practice.

Using a value of $t = 1$, the minimum value of n for the honest-but-curious case is $n = 3$, whilst the minimum value of n for the malicious case is $n = 4$. We found the following run-times for the above protocol. We ignore the time for the point multiplication step of the protocol as this is done outside the VIFF system and is essentially the cost of a non-distributed key generation centre. Hence, the following times denote the extra cost required for performing distributed key generation as opposed to centralised generation. All times are given in milli-seconds and count *wall-time*, hence it includes the time needed for data transmission etc.

Semi-Honest $n = 3$	Semi-Honest $n = 4$	Malicious $n = 4$
3 ms	4 ms	6 ms

The machines used in the benchmarks were equipped with hyper-threaded Intel Xeon CPUs (3.06 GHz clock speed) and 1 GiB of RAM. They were connected with a fast LAN, which is not typical of a *real world* installation but does give a lower bound on the resources required. Our computation time does not include the final local elliptic curve point multiplication. Hence the times represent the minimum *extra* time per key generation that are required by the servers to deal with a key generation request, compared with the case where a single centralised server is used.

All results are the average time for one inversion when executing a bulk computation with 100 inversions in parallel. Doing many operations in parallel is important in order to minimise the impact of network latencies, otherwise the latency will dominate the time. If only a single inversion is done, the time per inversion roughly doubles.

As can be seen from our timings the overall extra time needed to compute each identity based secret key is relatively small and would be easily accommodated in a deployed system, for example one could easily accommodate upto one million key extraction requests per day. We conclude that distributed key generation for Sakai–Kasahara style keys requires a relatively straight forward application of standard techniques from multi-party computation.

6 Extensions

We end the paper by outlining two extensions to the methods introduced in this paper.

6.1 Interactive Threshold Signature Scheme

We note that our techniques allow one to distribute the signing phase of the Zhang et al. signature scheme [25]. The public key of this scheme is $R = xP_2$, and to sign a message one computes

$$S = \left[\frac{1}{x + H(m)}\right] P_1$$

for some hash function $H : \{0,1\}^* \to \mathbb{F}_q$. Verification is performed by checking whether

$$\hat{t}(S, R + [H(m)]P_2) = \hat{t}(P_1, P_2).$$

Note, that we have reversed the groups in this scheme compared to the Sakai–Kasahara scheme so as to obtain shorter signatures. Each signature is thus given by an element of $\mathbb{G}_1$, and verification requires only one pairing computation. It is clear that our techniques for distributed key generation of Sakai–Kasahara keys also apply to distributing the signing operation for this signature scheme.

6.2 Distributed BB-2 Key Generation

Our techniques apply to many exponent-inversion key generation procedures, the most notable other system in this family is the BB-2 scheme from [6]. The BB-2 key generation procedure is as follows: For a master public key of $(P_1, R_x, R_y) = (P_1, xP_1, yP_1)$, with master secret key $(x, y) \in \mathbb{F}_q \times \mathbb{F}_q$, the key extraction phase is given by $S_{\mathrm{ID}} = (r, K_{\mathrm{ID}})$, where $r \in \mathbb{F}_q$ is chosen at random and

$$K_{\mathrm{ID}} = \left[\frac{1}{\mathrm{ID} + x + r \cdot y}\right] P_2.$$

We note that our techniques easily extend to this situation by the following distributed extraction algorithm:

- Using the PRSS property the servers obtain a sharing (r_i) of a random integer r.
- The servers compute a sharing (s_i) of $s = r \cdot y$ using the distributed multiplication protocol.
- The servers locally compute a sharing (z_i) of $z = \mathrm{ID} + x + s$.
- The servers then invert z using the inversion method of [1], to obtain a sharing (w_i).
- The servers then output (r_i) and $[w_i]P_2$.

This clearly requires two multiplication protocols, as opposed to one for the Sakai–Kasahara keys. The requesting user can recover r from (r_i) using the standard recovery method for Shamir secret sharing, and the value of K_{ID} from $[w_i]P_2$ via the method described previously. Note, that BB-2 keys are also self-checking since we must have

$$\hat{t}([\mathrm{ID}]P_1 + R_x + [r]R_y, K_{\mathrm{ID}}) = \hat{t}(P_1, P_2).$$

Hence, our self-checking procedure for preventing malicious servers responding with invalid keys will still apply.

Acknowledgements

The authors would like to thank Andy Dancer of Trend Micro Ltd for suggesting the problem, and Dario Catalano for providing a pointer to the literature. The authors would like to thank the EU FP7 project CACE. The second author was supported by a Royal Society Wolfson Merit Award.

References

1. Bar-Ilan, J., Beaver, D.: Non-cryptographic fault-tolerant computing in a constant number of rounds on interaction. In: Principles of Distributed Computing – PODC 1989, pp. 201–209. ACM Press, New York (1989)
2. Barreto, P.S.L.M., Libert, B., McCullagh, N., Quisquater, J.-J.: Efficient and provably-secure identity-based signatures and signcryption from bilinear maps. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 515–532. Springer, Heidelberg (2005)
3. Barreto, P.S.L.M., Naehrig, M.: Pairing-friendly elliptic curves of prime order. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 319–331. Springer, Heidelberg (2006)
4. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: 20th STOC, pp. 1–10 (1988)
5. Bogetoft, P., Christensen, D.L., Damgård, I., Geisler, M., Jakobsen, T., Krøigaard, M., Nielsen, J.D., Nielsen, J.B., Nielsen, K., Pagter, J., Schwartzbach, M., Toft, T.: Multiparty computation goes live. Cryptology ePrint Archive, Report 2008/068 (2008), `http://eprint.iacr.org/`
6. Boneh, D., Boyen, X.: Efficient selective-ID secure identity based encryption without random oracles. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 223–238. Springer, Heidelberg (2004)
7. Boneh, D., Boyen, X., Halevi, S.: Chosen ciphertext secure public key threshold encryption without random oracles. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 226–243. Springer, Heidelberg (2006)
8. Boneh, D., Franklin, M.: Identity based encryption from the Weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001)
9. Boyen, X.: General ad-hoc encryption from exponent inversion IBE. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 394–411. Springer, Heidelberg (2007)
10. Chaum, D., Crepeau, C., Damgård, I.: Multiparty unconditionally secure protocols. In: 20th STOC, pp. 11–19 (1988)
11. Chen, L., Cheng, Z.: Security proof of Sakai–Kasahara's identity-based encryption scheme. In: Smart, N.P. (ed.) Cryptography and Coding 2005. LNCS, vol. 3796, pp. 442–459. Springer, Heidelberg (2005)
12. Chen, L., Cheng, Z., Malone-Lee, J., Smart, N.P.: Efficient ID-KEM based on the Sakai–Kasahara key construction. IEE Proceedings - Information Security 153, 19–26 (2006)
13. Cramer, R., Damgård, I., Maurer, U.: General secure multiparty computation from any linear secret sharing scheme. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 316–334. Springer, Heidelberg (2000)

14. Cramer, R., Damgård, I., Nielsen, J.B.: Multiparty computation, an introduction. Manuscript, May 25th (2008)
15. Damgård, I., Geisler, M., Krøigaard, M., Nielsen, J.B.: Asynchronous multiparty computation: Theory and implementation. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 160–179. Springer, Heidelberg (2009)
16. Gennaro, R., Rabin, M., Rabin, T.: Simplified VSS and fast-track multi-party computation with applications to threshold cryptography. In: Principles of Distributed Computing – PODC 1989, pp. 101–111. ACM Press, New York (1998)
17. Sakai, R., Ohgishi, K., Kasahara, M.: Cryptosystems based on pairing over elliptic curve (in Japanese). In: The 2001 Symposium on Cryptography and Information Security, Oiso, Japan (January 2001)
18. Sakai, R., Kasahara, M.: ID based cryptosystems with pairing on elliptic curve. Cryptology ePrint Archive, Report 2003/054 (2003)
19. SCET: Secure Computing Economy and Trust. Homepage: `http://sikkerhed.alexandra.dk/uk/projects/scet`
20. Shamir, A.: How to share a secret. Communications of the ACM 22, 612–613 (1979)
21. Shamir, A.: Identity-based cryptosystems and signature schemes. In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 47–53. Springer, Heidelberg (1985)
22. Shoup, V.: Practical threshold signatures. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 207–220. Springer, Heidelberg (2000)
23. SIMAP: Secure Information Management and Processing. Homepage: `http://sikkerhed.alexandra.dk/uk/projects/simap`
24. VIFF: Virtual Ideal Functionality Framework. Homepage: `http://viff.dk/`
25. Zhang, F., Safavi-Naini, R., Susilo, W.: An efficient signature scheme from bilinear pairings and its applications. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 277–290. Springer, Heidelberg (2004)