Homomorphic encryption and secure comparison

Damgard, Ivan; Geisler, M.; Kroigaard, M.

Published in:
International Journal of Applied Cryptography

DOI:
10.1504/IJACT.2008.017048

Published: 01/01/2008

Document Version
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Citation for published version (APA):
Damgard, I., Geisler, M., & Kroigaard, M. (2008). Homomorphic encryption and secure comparison. International Journal of Applied Cryptography, 1(1), 22-31. DOI: 10.1504/IJACT.2008.017048

Homomorphic encryption and secure comparison

Ivan Damgård,* Martin Geisler and Mikkel Krøigård

BRICS, Department of Computer Science,
University of Aarhus, Denmark
E-mail: ivan@daimi.au.dk
E-mail: mg@daimi.au.dk
E-mail: mk@daimi.au.dk
*Corresponding author

Abstract: We propose a protocol for secure comparison of integers based on homomorphic encryption. We also propose a homomorphic encryption scheme that can be used in our protocol, makes it more efficient than previous solutions, and can also be used as the basis of efficient and general secure Multiparty Computation (MPC). We show how our comparison protocol can be used to improve security of online auctions, and demonstrate that it is efficient enough to be used in practice. For comparison of 16 bits numbers with security based on 1024 bits RSA (executed by two parties), our implementation takes 0.28 sec including all computation and communication. Using precomputation, one can save a factor of roughly 10.

Keywords: secure integer comparison; secure multiparty computation; MPC; homomorphic encryption; benchmark; online auctions; security; protocols; secret sharing.

Reference to this paper should be made as follows: Damgård, I., Geisler, M. and Krøigård, M. (2008) 'Homomorphic encryption and secure comparison, *Int. J. Applied Cryptography*, Vol. 1, No. 1, pp. 22–31.

Biographical notes: Ivan Damgård is a Professor at the Department of Computer Science, University of Aarhus, Denmark. His main research interests are cryptography, data security, discrete math, quantum information, algorithms and complexity. Currently, he is working on the Secure Information Management and Processing (SIMAP) project.

Martin Geisler is presently pursuing a PhD degree at the Department of Computer Science, University of Aarhus, Denmark. He began PhD study in 2006. His primary interests are protocols for secure multiparty computation and how to implement them elegantly and efficiently. Currently, he is working on the Secure Information Management and Processing (SIMAP) project.

Mikkel Krøigård is a PhD student at the Department of Computer Science, University of Aarhus, Denmark. He began his PhD in 2006 with a focus on developing efficient protocols for secure multiparty computation. Currently, he is working on the Secure Information Management and Processing (SIMAP) project.

1 Introduction

Secure comparison of integers is the following problem: two or more players are given integers n_A, n_B, where one or both are private, that is, not known to all players. We then want to decide whether $n_A \geq n_B$, while making sure that the comparison result is the only new information that becomes known. Many variants of this problem exist depending on whether n_A, n_B are known to particular players, or unknown to everyone. It may even be the case that the comparison result is not supposed to be public, but is to be produced in encrypted form, for instance.

Secure comparison is a special case of general secure computation where all players hold private inputs and want to compute some agreed function of these inputs. Comparison protocols are very important ingredients in many potential applications of secure computation. Examples of this include auctions, benchmarking and secure extraction of statistical data from databases.

As a concrete example to illustrate the application of the results from this paper, we take a closer look at online auctions: Many online auction systems offer as a service to their customers that one can submit a maximum bid to the system. It is then not necessary to be online permanently, the system will automatically bid for you, until you win the auction or your specified maximum is exceeded. We assume in the following what we believe is a realistic scenario, namely that the auction system needs to handle bidders that bid online manually, as well as others that use the option of submitting a maximum bid.

Clearly, such a maximum bid is confidential information: both the auction company and other participants in the auction have an interest in knowing such maximum bids in advance, and could exploit such knowledge to their advantage: The auction company could force higher prices (what is known as 'shill bidding') and thereby increase its income and other bidders might learn how valuable a given item is to others and change their strategy accordingly.

In a situation where anyone can place a bid by just connecting to a website, the security one can obtain by storing the maximum bids with a single trusted party is questionable, in particular if that trusted party is the auction company. Indeed, there are cases known from real auctions where an auction company has been accused of misusing its knowledge of maximum bids (Rode, 2004).

An obvious solution is to share the responsibility of storing the critical data among several parties, and do the required operations via secure computation. One can then make sure that, unless all parties are corrupted, every time the bid goes up, it will become known whether a given player is still in the game because his maximum is larger then the current price, but the actual value of the maximum will remain secret. To keep the communication pattern simple and to minimise problems with maintenance and other logistical problems, it seems better to keep the number of involved players small. We therefore consider the following model.

An input client C supplies an ℓ bit integer m as private input to the computation, which is done by players A and B. Because of our motivating scenario, we require that the input is supplied by sending one message to A, respectively to B, and no further interaction with C is necessary. One may, for instance, think of A as the auction house and B as an accounting company. We will also refer to these as the *server* and *assisting server*.

An integer x (which we think of as the currently highest bid) is public input. As public output, we want to compute one bit that is 1 if $m > x$ and 0 otherwise, that is, the output tells us if C is still in the game and wants to raise the bid, say by some fixed amount agreed in advance. Of course, we want to do the computation securely so that neither A nor B learns any information on m other than the comparison result.

We will assume that players are honest but curious. We believe this is quite a reasonable assumption in our scenario: C may submit incorrectly formed input, but since the protocol handles even malformed input deterministically, he cannot gain anything from this: any malformed bid will determine a number x_0 such that when the current price reaches x_0, the protocol output will cause C to leave the game. So this is equivalent to submitting x_0 in correct format. Moreover, the actions of A and B can be checked after the auction is over – if C notices that incorrect decisions were taken, he can prove that his bid was not correctly handled. Such 'public disgrace' is likely to be enough to discourage cheating in our scenario. Nevertheless, we sketch later in this paper how to obtain active security at moderate extra cost.

Some of the results in this paper were presented in preliminary form at the ACISP 2007 conference.

1.1 Our contribution

In this paper, we first propose a new homomorphic cryptosystem that is well suited for our application, this is the topic of Section 2. The cryptosystem is much more efficient than, for example, the encryption scheme by (Paillier 1999) in terms of en- and decryption time. The efficiency is obtained partly by using a variant of Groth's idea of exploiting subgroups of $\mathbb{Z}_n^*$ for an RSA modulus n (Groth, 2005), and partly by aiming for a rather small plaintext space, of size $\theta(\ell)$.

Later in this paper, show how the size of the plaintext space can be extended to allow for more general applications, how to do threshold decryption, and how to apply these observations to do secure Multiparty Computation (MPC).

In Section 3 we propose a comparison protocol in our model described above, based on additive secret sharing and homomorphic encryption. The protocol is a new variant of an idea originating in a paper by Blake and Kolesnilov (2004). Their original idea was also based on homomorphic encryption but required a plaintext space of size exponential in ℓ. Here, we present a new technique allowing us to make do with a smaller plaintext space. This means that the exponentiations we do will be with smaller exponents and this improves efficiency. Also, we save computing time by using additive secret sharing as much as possible instead of homomorphic encryption.

As mentioned, our encryption is based on a k bit RSA modulus. In addition there is an 'information theoretic' security parameter t involved which is approximately the logarithm of the size of the subgroup of $\mathbb{Z}_n^*$ we use. Here, t needs to be large enough so that exhaustive search for the order of the subgroup and other generic attacks are not feasible. Section 4 contains more information about the security of the protocol.

In the protocol, C sends a single message to A and another to B, both of size $\mathcal{O}(\ell \log \ell + k)$ bits. To do the comparison, there is one message from A to B and one from B to A. The size of each of these messages is $\mathcal{O}(\ell k)$ bits. As for computational complexity, both A and B need to do $\mathcal{O}(\ell(t + \log \ell))$ multiplications mod n. Realistic values of the parameters might be $k = 1024$, $t = 160$ and $\ell = 16$. In this case, counting the actual number of multiplications works out to roughly 7 full scale exponentiations mod n, and takes 0.28 sec in our implementation, including all computation and communication time. Moreover, most of the work can be done as preprocessing. Using this possibility in the concrete case above, the online work for B is about 0.6 exponentiations for A and 0.06 for B, so that we can expect to save a factor of at least 10 compared to the basic version. It is clear that the online performance of such a protocol is extremely important: auctions often run up a certain deadline, and bidders in practice sometimes play a strategy where they suddenly submit a much larger bid just before the deadline in the hope of taking other bidders by surprise. In such a scenario, one cannot wait a long time for a comparison protocol to finish.

We emphasise that, while it may seem easier to do secure comparison when one of the input numbers is public, we do this variant only because it comes up naturally in our example scenario. In fact, it is straightforward to modify our protocol to fit related scenarios. For instance, the case where A has a private integer a, B has a private integer b and we want to compare a and b, can be handled with essentially the same cost as in our model. Moreover, at the expense of a factor about 2 in the round, communication and computational complexities, our protocol generalises to handle comparison of two integers that are shared between

A and B, that is, are unknown to both of them. It is also possible to keep the comparison result secret, that is, produce it in encrypted form. More details on this are given in Section 5.

Finally, in Section 6 we describe our implementation and the results of a benchmark between our proposed protocol and the one from Fischlin (2001).

1.2 Related work

There is a very large amount of work on secure auctions, which we do not attempt to survey here, as our main concern is secure protocols for comparison, and the online auction is mainly a motivating scenario. One may of course do secure comparison of integers using generic MPC techniques. For the two-party case, the most efficient generic solution is based on Yao-garbled circuits, which were proposed for use in auctions by Naor et al. (1999). Such methods are typically less efficient than ad hoc methods for comparison – although the difference is not very large when considering passive security. For instance, the Yao garbled circuit method requires – in addition to garbling the circuit – that we do an oblivious transfer of a secret key for every bit position of the numbers to compare. This last part is already comparable to the cost of the best known ad hoc methods.

There are several existing ad hoc techniques for comparison, we already mentioned the one from Blake and Kolesnikov (2004) above, a later variant appeared in Blake and Kolesnikov (2006), allowing comparison of two numbers that are unknown to the parties. A completely different technique was proposed earlier by Fischlin (2001).

It should be noted that previous protocols typically are for the model where A has a private number a, B has a number b, and we want to compare a and b. Our model is a bit different, as we have one public number that is to be compared to a number that should be known to neither party, and so has to be shared between them. However, the distinction is not very important: previous protocols can quite easily be transformed to our model, and as mentioned above, our protocol can also handle the other models at marginal extra cost. Therefore the comparison of our solution to previous work can safely ignore the choice of model.

Fischlin's protocol is based on the well-known idea of encrypting bits as quadratic residues and non-residues modulo an RSA modulus, and essentially simulates a Boolean formula that computes the result of the comparison. Compared to Blake and Kolesnikov (2004, 2006), this saves computing time, since creating such an encryption is much faster than creating a Paillier encryption. However, in order to handle the non-linear operations required in the formula, Fischlin extends the encryption of each bit into a sequence of λ numbers, where λ is a parameter controlling the probability that the protocol returns an incorrect answer. Since these encryptions have to be communicated, we get a communication complexity of $\Omega(\lambda\ell k)$ bits. The parameter λ should be chosen such that $5\ell 2^{-\lambda}$ is an acceptable (small enough) error probability, so this makes the communication complexity significantly larger than the $\mathcal{O}(\ell k)$ bits one gets in our protocol and the one from Blake and Kolesnikov (2006).

The computational complexity for Fischlin's protocol is $\mathcal{O}(\ell\lambda)$ modular multiplications, which for typical parameter values is much smaller than that of Blake and Kolesnikov (2004, 2006), namely $\mathcal{O}(\ell k)$ multiplications.[1] Fischlin's result is not directly comparable to ours, since our parameter t is of a different nature than Fischlin's λ: t controls the probability that the best known generic attack breaks our encryption scheme, while λ controls the probability that the protocol gives incorrect results. However, if we assume that parameters are chosen to make the two probabilities be roughly equal, then the two computational complexities are asymptotically the same.

Thus, in a nutshell, Blake and Kolesnikov (2004, 2006) have small communication and large computational complexity while Fischlin (2001) is the other way around. In comparison, our contribution allows us to get 'the best of both worlds'. In Section 6 we give results of a comparison between implementations of our own and Fischlin's protocols. Finally, note that our protocol always computes the correct result, whereas Fischlin's has a small error probability.

In concurrent independent work, Garay et al. (2007) propose protocols for comparison based on homomorphic encryption that are somewhat related to ours, although they focus on the model where the comparison result is to remain secret. They present a logarithmic round protocol based on emulating a new Boolean circuit for comparison, and they also have a constant round solution. In comparison, we do not consider the possibility of saving computation and communication in return for a larger number of rounds. On the other hand, their constant round solution is based directly on Blake and Kolesnikov's method, that is, they do not have our optimisation that allows us to make do with a smaller plaintext space for the encryption scheme, which means that our constant round protocol is more efficient.

2 Homomorphic encryption

For our protocol we need a semantically secure and additively homomorphic cryptosystem which we will now describe.

To generate keys, we take as input parameters k, t and ℓ, where $k > t > \ell$. We first generate a k bit RSA modulus $n = pq$ for primes p, q. This should be done in such a way that there exists another pair of primes u, v, both of which should divide $p-1$ and $q-1$. We will later be doing additions of small numbers in $\mathbb{Z}_u$ where we want to avoid reductions modulo u, but for efficiency we want u to be as small as possible. For these reasons we choose u as the minimal prime greater than $\ell + 2$. The only condition on v is that it is a random t bit prime.

Finally, we choose random elements $g, h \in \mathbb{Z}_n^*$ such that the multiplicative order of h is v modulo p and q and g has order uv. The public key is now $pk = (n, g, h, u)$ and the secret key is $sk = (p, q, v)$. The plaintext space is $\mathbb{Z}_u$, while the ciphertext space is $\mathbb{Z}_n^*$.

To encrypt $m \in \mathbb{Z}_u$, we choose r as a random $2t$ bit integer, and let the ciphertext be

$$E_{pk}(m, r) = g^m h^r \bmod n$$

We note that by choosing r as a much larger number than v, we make sure that h^r will be statistically indistinguishable from a uniformly random element in the group generated by h. The owner of the secret key (who knows v) can do it more efficiently by using a random $r \in \mathbb{Z}_v$.

For decryption of a ciphertext c, it turns out that for our main protocol, we will only need to decide whether c encrypts 0 or not. This is easy, since $c^v \bmod n = 1$ if and only if c encrypts 0. This follows from the fact that v is the order of h, uv is the order of g and $m < u$. If the party doing the decryption has also stored the factors of n, one can optimise this by instead checking whether $c^v \bmod p = 1$, which will save a factor of 3–4 in practice.

It is also possible to do a 'real' decryption by noting that

$$E_{pk}(m, r)^v = (g^v)^m \bmod n$$

Clearly, g^v has order u, so there is a 1–1 correspondence between values of m and values of $(g^v)^m \bmod n$. Since u is very small, one can simply build a table containing values of $(g^v)^m \bmod n$ and corresponding values of m.

To evaluate the security, there are various attacks to consider: factoring n will be sufficient to break the scheme, so we must assume factoring is hard. Also note that it does not seem easy to compute elements with orders such as g, h unless you know the factors of n, so we implicitly assume here that knowledge of g, h does not help to factor. Note that it is very important that g, h both have the same order modulo both p and q. If g had order uv modulo p but was 1 modulo q, then g would have the correct order modulo n, but $\gcd(g-1, n)$ would immediately give a factor of n. One may also search for the secret key v, and so t needs to be large enough so that exhaustive search for v is not feasible. A more efficient generic attack (which is the best we know of) is to compute $h^R \bmod n$ for many large and random values of R. By the 'birthday paradox', we are likely to find values R, R' where $h^R = h^{R'} \bmod n$ after about $2^{t/2}$ attempts. In this case v divides $R - R'$, so generating a few of these differences and computing the greatest common divisor will produce v. Thus, we need to choose t such that $2^{t/2}$ exponentiations is infeasible.

To say something more precise about the required assumption, let G be the group generated by g, and H the group generated by h. We have $H \leq G$ and that a random encryption is simply a uniformly random element in G. The assumption underlying security is now

Conjecture 1: *For any constant ℓ and for appropriate choice of t as a function of the security parameter k, the tuple (n, g, h, u, x) is computationally indistinguishable from (n, g, h, u, y), where n, g, h, u are generated by the key generation algorithm sketched above, x is uniform in G and y is uniform in H.*

Proposition 1: *Under the above conjecture, the cryptosystem is semantically secure.*

Proof: Consider any polynomial time adversary who sees the public key, chooses a message m and gets an encryption of m, which is of the form $g^m h^r \bmod n$, where g has order uv and h has order v modulo p and q. The conjecture now states that even given the public key, the adversary cannot distinguish between a uniformly random element from H and one from G. But h^r was already statistically indistinguishable from a random element in H, and so it must also be computationally indistinguishable from a random element in G. But this means that the adversary cannot distinguish the entire encryption from a random element of G, and this is equivalent to semantic security – recall that one of the equivalent definitions of semantic security requires that encryptions of m be computationally indistinguishable from random encryptions. □

The only reason we set t to be a function of k is that the standard definition of semantic security talks about what happens asymptotically when a *single* security parameter goes to infinity. From the known attacks sketched above, we can choose t to be much smaller than k. Realistic values might be $k = 1024, t = 160$.

A central property of the encryption scheme is that it is homomorphic over u, that is,

$$E_{pk}(m, r)E_{pk}(m', r') \bmod n = E_{pk}(m + m' \bmod u, r + r').$$

The cryptosystem is related to that of Groth (2005), in fact ciphertexts in his system also have the form $g^m h^r \bmod n$. The difference lies in the way n, g and h are chosen. In particular, our idea of letting h, g have the same order modulo p and q allows us to improve efficiency by using subgroups of Z_n^* that are even smaller than those from Groth (2005).

3 Comparison protocol

For the protocol, we assume that A has generated a key pair $sk = (p, q, v)$ and $pk = (n, u, g, h)$ for the homomorphic cryptosystem we described previously. The protocol proceeds in two phases: an input sharing phase in which the client must be online, and a computation phase where the server and assisting server determine the result while the client is off-line. See Figure 1 for an overview.

In the input sharing phase C secret shares his input m between A and B:

- Let the binary representation of m be $m_\ell, \ldots, m_1$, where m_1 is the least significant bit. C chooses, for $i = 1, \ldots, \ell$, random pairs $a_i, b_i \in \mathbb{Z}_u$ subject to $m_i = a_i + b_i \bmod u$.
- C sends privately $a_1, \ldots, a_\ell$ to A and $b_1, \ldots, b_\ell$. This can be done using any secure public-key cryptosystem with security parameter k, and requires communicating $O(\ell \log \ell + k)$ bits.[2] In practice, a standard SSL connection would probably be used.

In the second phase we wish to determine the result $m > x$ where x is the current public price (with binary representation $x_\ell, \ldots, x_1$).

Assuming a value $y \in \mathbb{Z}_u$ has been shared additively between A and B, as C did it in the first phase, we write $[y]$ for the pairs of shares involved, so $[y]$ stands for 'a sharing of' y. Since the secret sharing scheme is linear over $\mathbb{Z}_u$, A and B can compute from $[y]$, $[w]$ and a publically known value α a sharing $[y + \alpha w \bmod u]$. Note that this does not require interaction but merely local computation. The protocol proceeds as follows:

- A and B compute, for $i = 1, \ldots, \ell$ sharings $[w_i]$ where

$$w_i = m_i + x_i - 2x_i m_i = m_i \oplus x_i$$

- A and B now compute, for $i = 1, \dots, \ell$ sharings $[c_i]$ where

$$c_i = x_i - m_i + 1 + \sum_{j=i+1}^{\ell} w_j$$

  Note that if $m > x$, then there is exactly one position i where $c_i = 0$, otherwise no such position exists. Note also, that by the choice of u, it can be seen that no reductions modulo u take place in the above computations.

- Let α_i and β_i be the shares of c_i that A and B have now locally computed. A computes encryptions $E_{pk}(\alpha_i, r_i)$ and sends them all to B.

- B chooses at random $s_i \in \mathbb{Z}_u^*$ and s_i' as a $2t$ bit integer and computes a random encryption of the form

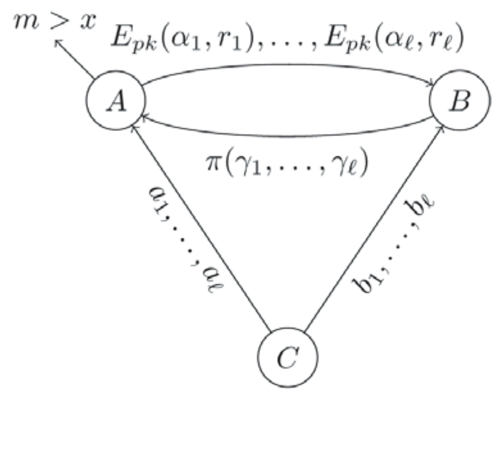
$$\gamma_i = (E_{pk}(\alpha_i, r_i) \cdot g^{\beta_i})^{s_i} \cdot h^{s_i'} \bmod n$$

  Note that, if $c_i = 0$, this will be an essentially random encryption of 0, otherwise it is an essentially random encryption of a random non-zero value.

  B sends these encryptions to A in randomly permuted order.

- A uses his secret key to decide, as described in the previous section, whether any of the received encryptions contain 0. If this is the case, he outputs '$m > x$', otherwise he outputs '$m \le x$'.

Figure 1 Our proposed protocol with both phases illustrated. In the first phase C sends shares to A and B. The second phase consists of a message from A to B and a reply, which A can decrypt to learn the result of the computation


$m > x$
$E_{pk}(\alpha_1, r_1), \dots, E_{pk}(\alpha_\ell, r_\ell)$
A
B
$\pi(\gamma_1, \dots, \gamma_\ell)$
$a_1, \dots, a_\ell$
$b_1, \dots, b_\ell$
C

A note on preprocessing: one can observe that the protocol frequently instructs players to compute a number of form $h^r \bmod n$ where r is randomly chosen in some range, typically $[0 \dots 2^{2t}]$. Since these numbers do not depend on the input, they can be precomputed and stored. As mentioned in the introduction, this has a major effect on performance because all other exponentiations are done with very small exponents.

4 Security

In this section the protocol is proven secure against an honest but curious adversary corrupting a single player at the start of the protocol.

The client C has as input its maximum bid m and all players have as input the public bid x. The output given to A is the evaluation of $m > x$, and B and C get no output.

In the following we argue correctness and we argue privacy using a simulation argument. This immediately implies that our protocol is secure in Canetti's model for secure function evaluation (Canetti, 2000) against a static and passive adversary.

4.1 Correctness

The protocol must terminate with the correct result: $m > x \iff \exists i : c_i = 0$. This follows easily by noting that both $x_i - m_i + 1$ and w_i is non-negative so

$$c_i = 0 \iff x_i - m_i + 1 + \sum_{j=i+1}^{\ell} w_j = 0$$
$$\iff x_i - m_i + 1 = 0 \wedge \sum_{j=i+1}^{\ell} w_j = 0$$

We can now conclude correctness of the protocol since $x_i - m_i + 1 = 0 \iff m_i > x_i$ and $\sum_{j=i+1}^{\ell} w_j = 0 \iff \forall j > i : m_j = x_j$, which together imply $m > x$. Note that since the sum of the w_j is positive after the first position in which $x_i \neq m_i$, there can be at most one zero among the c_i.

4.2 Privacy

Privacy in our setting means that A learns only the result of the comparison and B learns nothing new. We can ignore the client as it has the only secret input and already knows the result based on its input.

First assume that A is corrupt, that is, that A tries to deduce information about the maximum bid based on the messages it sees. From the client, A sees both his own shares $a_1, \dots, a_\ell$, and the ones for B encrypted under some semantically secure cryptosystem, for example, SSL. From B, A sees the message:

$$\left(E_{pk}(\alpha_i, r_i) \cdot g^{\beta_i}\right)^{s_i} \cdot h^{s_i'} \bmod n$$

By the homomorphic properties of our cryptosystem this can be rewritten as

$$E_{pk}(s_i \cdot \alpha_i, s_i \cdot r_i) \cdot E_{pk}(s_i \cdot \beta_i, s_i')$$
$$= E_{pk}(s_i(\alpha_i + \beta_i), s_i \cdot r_i + s_i')$$

In order to prove that A learns no additional information, we can show that A could – given knowledge of the result, the publically known number and nothing else – simulate the messages it would receive in a real run of the protocol.

The message received and seen from the client can trivially be simulated as it consists simply of ℓ random numbers modulo u and ℓ encrypted shares. The cryptosystem used for these messages is semantically secure, so the encrypted shares for B can be simulated with encryptions of random numbers.

To simulate the messages received from B, we use our knowledge of the result of the comparison. If the result is '$m > x$', we can construct the second message as $\ell - 1$ encryptions of a non-zero element of $\mathbb{Z}_u^*$ and one encryption of zero in a random place in the sequence. If the result is '$m \leq x$', we instead construct ℓ encryptions of non-zero elements in $\mathbb{Z}_u^*$.

If we look at the encryptions that B would send in a real run of the protocol, we see that the plaintexts are of form $(\alpha_i + \beta_i)s_i \bmod u$. Since s_i is uniformly chosen, these values are random in Z_u if $\alpha_i + \beta_i \neq 0$ and 0 otherwise. Thus the plaintexts are distributed identically to what was simulated above. Furthermore, the ciphertexts are formed by multiplying $g^{(\alpha_i+\beta_i)s_i}$ by

$$h^{s_i r_i + s_i'} = h^{s_i r_i} h^{s_i'}$$

But h has order v which is t bits long, and therefore taking h to the power of the random $2t$ bit number s_i' will produce something which is statistically indistinguishable from the uniform distribution on the subgroup generated by h. But since $h^{s_i r_i} \in \langle h \rangle$, the product will indistinguishable from the uniform distribution on $\langle h \rangle$. So the s_i' effectively mask out $s_i r_i$ and makes the distribution of the encryption statistically indistinguishable from a random encryption of $(\alpha_i + \beta_i)s_i$. Therefore, the simulation is statistically indistinguishable from the real protocol messages.

The analysis for the case where B is corrupt is similar. Again we will prove that we can simulate the messages of the protocol. The shares received from the client and the encryptions seen are again simply ℓ random numbers modulo u and ℓ random encryptions and are therefore easy to simulate. Also, B receives the following from A:

$$E_{pk}(\alpha_i, r_i)$$

But since the cryptosystem is semantically secure, we can make our own random encryptions instead and their distribution will be computationally indistinguishable from the one we would get by running the protocol normally.

5 Extensions

Although the protocol and underlying cryptosystem presented in this paper are specialised to one kind of comparison, both may be extended. In this section we will first consider how the protocol can be modified to handle more general comparisons where one input is not publically known, and we will also sketch how active security can be achieved. We also consider application of the cryptosystem to general MPC.

5.1 Both inputs are private

Our protocol extends in a straightforward way to the case where A and B have private inputs a, b and we want to compare them. In this case, A can send to B encryptions of the individual bits of a, using his own public key. Since the cryptosystem is homomorphic over u, B can now do the linear operations on the bits of a and b that in the original protocol were done on the additive shares of the bits. Note that B has his own input in cleartext, so the encryptions of the exclusive-or of bits in a and b can be computed without interaction, using the formula $x \oplus y = x + y - 2xy$ which is linear if one of x, y is a known constant. B can therefore produce, just as before, a set of encryptions of either random values or a set that contains a single 0. These are sent to A for decryption. The only extra cost of this protocol compared to the basic variant above is that B must do $\mathcal{O}(l)$ extra modular multiplications, and this is negligible compared to the rest of the protocol.

5.2 Both inputs are shared, secret output

The case where both numbers a, b to compare are unknown to A and B can also be handled. Assume both numbers are shared between A and B using additive shares. The only difficulty compared to the original case is the computation of shares in the exclusive-or of bits in a and b. When all bits are unknown to both players, this is no longer a linear operation. But from the formula $x \oplus y = x + y - 2xy$, it follows that it is sufficient to compute the product of two shared bits securely. Let x, y be bits that are shared so $x = x_a + x_b \bmod u$ and $y = y_a + y_b \bmod u$, where A knows x_a, y_a and B knows x_b, y_b. Now, $xy = x_a y_a + x_b y_b + x_b y_a + x_a y_b$. The two first summands can be computed locally, and for, for example, $x_a y_b$, A can send to B an encryption $E_{pk}(x_a)$. B chooses $r \in Z_u$ at random and computes an encryption $E_{pk}(x_a y_b - r \bmod u)$ using the homomorphic property. This is sent to A, and after decryption $(x_a y_b - r \bmod u, r)$ forms a random sharing of $x_a y_b$. This allows us to compute a sharing of xy and hence of $x \oplus y$. Putting this method for computing exclusive-ors together with the original protocol, we can do the comparison at cost roughly twice that of the original protocol.

It follows from an observation in Toft (2007) that a protocol comparing shared inputs that gives a public result can always be easily transformed to one that gives the result in shared form so it is unknown to both parties. The basic idea is to first generate a shared random bit $[B]$ where B is unknown to both parties. Then from (bit-wise) shared numbers a, b, we compute two new bit-wise shared numbers $c = a + (b - a)B, d = b + (a - b)B$, this just requires a linear number of multiplications. Note that $(c, d) = (a, b)$ if $B = 0$ and $(c, d) = (b, a)$ otherwise. Finally, we compare c, d and get a public result B'. The actual result can then be computed in shared form as $[B \oplus B']$.

5.3 Active security

Finally, we sketch how one could make the protocol secure against active cheating. For this, we equip both A and B with private/public key pairs (sk_A, pk_A) and (sk_B, pk_B) for our cryptosystem. It is important that both key pairs are constructed with the *same* value for u. The client C will now share its input as before, but will in addition send to both players encryptions of all of A's shares under pk_A and all of B's shares under pk_B. Both players are now committed to their shares, and can therefore prove in zero-knowledge during the protocol that they perform correctly. Since the cryptosystem is homomorphic and the secret is stored in the exponent, one can use standard protocols for proving relations among discrete logs, see for instance (Brands, 1997; Fujisaki and Okamoto, 1997; Schnorr, 1991). Note that since the two public keys use the same value of u, it is possible to prove relations involving both public keys, for instance, given $E_{pk_A}(x)$ and $E_{pk_B}(y)$, that $x = y$. In the final stage, B must show that a set of values encrypted under pk_A is a permutation of a set of committed values. This is known as the shuffle problem and many efficient solutions for this are known – see, e.g. Groth (2003). Overall, the cost of adding active security will therefore be quite moderate, but the computing the exact cost requires further work: The type of protocol we would use to check players' behaviour typically have error probability 1 divided by the smallest prime factor in the order of the group used. This would be $1/u$ in our case, and the protocols will have to be repeated if $1/u$ is not sufficiently small. This results in a trade off: we want a small u to make the original passively secure protocol more efficient, but a larger value of u makes the protocols we use for checking players' behaviour more efficient. An exact analysis of this is outside the scope of this paper.

5.4 *Using the cryptosystem for MPC*

In Cramer et al. (2001) it is described how any homomorphic cryptosystem with certain properties may be used for general MPC, where several parties want to compute an arbitrary function of their inputs. The cryptosystem described in this paper was already shown to be homomorphic, but in order for us to use the results from Cramer et al. (2001), we need to establish some other properties of the cryptosystem.

The basic ideas in the construction from Cramer et al. (2001) are as follows. A public key pk is assumed to be set up initially, where the secret key sk has been secret shared among the players. We assume an adversary that can corrupt (only) certain sets of players and sk is shared such that no corruptible set has information on it, whereas the set of players following the protocol could reconstruct it from their shares. What is needed is now a protocol for so called threshold decryption, where the players (using their shares of sk as private input) can decrypt a ciphertext securely, that is, compute the plaintext while revealing no side information.

The cryptosystem is assumed to be additively homomorphic over some ring $\mathcal{R}$, in our case $\mathcal{R} = Z_v$. If we want to compute the value $f(x_1, ..., x_n)$ where the x_i's are private inputs and f is a given function, we think of the inputs and outputs as elements in $\mathcal{R}$, and we assume that f is given as an arithmetic circuit over $\mathcal{R}$, that is, computing f can be done by additions and multiplications in $\mathcal{R}$. This is without loss of generality as any Boolean circuit can be simulated with these operations. But if we can have $\mathcal{R} = Z_v$ for sufficiently large v, we may be able to compute f by a circuit over Z_v that is much smaller than a Boolean circuit. In particular, we can simulate integer addition and multiplication by single operations in Z_v if v is large enough compared to the inputs.

To compute f securely, players submit their inputs encrypted under pk and we then simulate the circuit computing f gate by gate. If we have encryptions $E_{pk}(a)$, $E_{pk}(b)$ of the inputs to an addition gate, the homomorphic property immediately implies that we get an encryption of the output $a + b$ as $E_{pk}(a) \cdot E_{pk}(b) = E_{pk}(a + b)$. For multiplication, a simple protocol is given in Cramer et al. (2001) for computing securely an encryption $E_{pk}(ab)$ from $E_{pk}(a)$, $E_{pk}(b)$. It requires a single threshold decryption and that each player sends a ciphertext to all other players. At end of the day, we have encryptions of the output values, which we then decrypt using threshold decryption.

To make this work against a passive adversary, all you need is the homomorphic property and secure threshold decryption. For active security, one needs in addition protocols by which players can show that they follow the rules during the multiplication operations.

We now look at how our encryption scheme can be made to fit this model: In our comparison protocol, we have assumed that the input was given as sharings or encryptions of single bits of the inputs. But as explained above, for general-purpose MPC, it is desirable to be able to supply an entire input number by sending a single ciphertext. We therefore need to consider the size of the our plaintext space. In the cryptosystem as described above, the plaintext space must be small in order to decrypt efficiently. Therefore, to have a larger plaintext space without losing efficiency, we propose the following modification to the cryptosystem.

Instead of u being a prime, we make u a power of 2. After raising a ciphertext $g^m h^r \bmod n$ to the power of v, we get the number $(g^v)^m \bmod n$, where $g^v \bmod n$ is of order 2^c for some integer c. This means that to get m, we need to solve the discrete log problem where the base is 2-smooth. We can apply the Pohlig-Hellman algorithm (Pohlig and Hellman, 1978) and solve the problem using $\mathcal{O}(c)$ modular multiplications.

It is now possible to use this cryptosystem for passive security MPC. It follows from the above that all we need is to show that secure threshold decryption is possible. So instead of having a single private key, we will instead secret-share it between the N parties. One way to do this is additive integer secret sharing, that is, the private share of player i, for $i = 1 \dots N$, is a random $2t$-bit number v_i, and we give to all players $v_0 = v - \sum_{i=1}^{N} v_i$. Note that even if all shares except one, say v_i, are known, the only information related to v that can be computed is $v - v_i$, which is statistically indistinguishable from a random (negative) $2t$-bit number.

Now the parties can work together to decrypt a ciphertext $c = g^m h^r \bmod n$. Each player simply broadcasts $c^{v_i} \bmod n$, and then everyone can compute

$$\prod_{i=0}^{N} c^{v_i} \bmod n = (g^v)^m \bmod n$$

If we include $g^v \bmod n$ in the public key, m can now be found as in the single user case above. Note that we now need to make a stronger assumption, namely that the cryptosystem remains semantically secure, even when $g^v \bmod n$ is given.

It is straightforward to show:

Lemma 1: *The above threshold decryption protocol is secure against a passive and static adversary, corrupting up to $N - 1$ players.*

Proof: To prove the security of the decryption protocol, we will show that given the public information, that is, public key as well as the ciphertext and plaintext, we can efficiently simulate everything else the adversary sees, namely the shares of corrupted players and the messages received from the honest players in the protocol.

Assume without loss of generality that player N is the only uncorrupted player. Now, generate shares $v_0, ..., v_N$ as described above using secret value 0, and give $v_0, ..., v_{N-1}$ to the adversary, to play the role of shares of corrupted players. As argued above, since the adversary does not see v_N, this has distribution statistically close to what would be the case if the real v-value had been used. We simulate the message sent by player N when decrypting c as

$$x = c^v \Big(\prod_{i=0}^{N-1} c^{v_i}\Big)^{-1}$$

In the real world, the v_i are chosen at random such that $\sum_{i=0}^{N} v_i = v$. It follows that x has the same distribution as player N's contribution in the real world. Note that c^v needs to be computed as well, but since we are given g^v along with the public key, we may compute c^v as $(g^v)^m = c^v$.

This shows that the decryption protocol can be simulated without knowledge of the key shares for the honest player, and therefore that the adversary gains no information about v by participating in the protocol.

It follows from all of the above that we may use our cryptosystem, with the modification for the plaintext space, to achieve general MPC that is secure against a passive and static adversary corrupting all but one of the players.
It is possible to have active security as well, according to Cramer et al. (2001) this only requires existence of honest verifier zero-knowledge proofs (Σ-protocols) for proving certain statements on ciphertexts. It follows from Fujisaki and Okamoto (1997) that such Σ-protocols can be constructed using standard methods (even though the orders of g and h are hidden). Unfortunately, soundness of the most efficient versions of the protocols from Fujisaki and Okamoto (1997) require that the order of g, h have only large prime factors. This is not true in our case. The problem can be solved at some cost in efficiency by using the basic protocols with 1-bit challenges and running several instances in parallel.

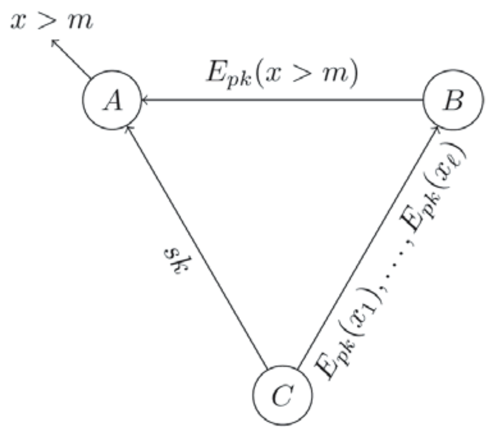
6 Complexity and performance

In this section we measure the performance of our solution through practical tests. The protocol by Fischlin (2001) provides a general solution to comparing two secret integers using fewer multiplications than the other known general solutions. We show that in the special case where one integer is publically known and the other is additively shared between two parties, our solution provides for faster comparisons than our adaptation of Fischlin (2001).

6.1 *Setup and parameters*

As described above, our special case consists of a server, an assisting server and a client. The client must be able to send his value and go off-line, whereafter the other two parties should be able to do the computations together. In our protocol the client simply sends additive shares to each of the servers and goes off-line. However, the protocol by Fischlin needs to be adapted to this scenario before we can make any reasonable comparisons. A very simple way to mimic the additive sharing is for the client to simply send his secret key used for the encoding of his value to the server while sending the actual encoding to the assisting server. Clearly the computations can now be done by the server and assisting server alone, where the server plays the role of the client. The modified protocol is shown in Figure 2.

Figure 2 The modified Fischlin protocol



Note: The client C can go off-line after having sent the key to A and the encryptions to B. From that point the protocol proceeds as in Fischlin (2001).

Together, the key and encoding determine the client's secret value, but the key or the encoding alone do not. The key of course reveals no information about the value. Because of semantic security, the encryption alone does not reveal the secret to a computationally bounded adversary.

Another issue is how to compare the two protocols in a fair way. Naturally, we want to choose the parameters such that the two protocols offer the same security level, but it is not clear what this should mean: some of the parameters in the protocols control events of very different nature. Below, we describe the choices we have made and the consequences of making different choices.

Both protocols use an RSA modulus for their encryption schemes, and it is certainly reasonable to use the same bit length of the modulus in both cases, say 1024 bits. Our encryption scheme also needs a parameter t which we propose to choose as $t = 160$. This is because the best known attack tries to have random results of exponentiations collide in the subgroup with about 2^{160} elements. Assuming the adversary cannot do much more than 2^{40} exponentiations, the collision probability is roughly $2^{2\cdot 40}/2^{160} = 2^{-80}$.

We do not have this kind of attack against Fischlin, but we do have an error probability of $5\ell 2^{-\lambda}$ per comparison. If we choose the rationale that the probability of 'something going wrong' should be the same in both protocols, we should choose λ such that Fischlin's protocol has an error probability of 2^{-80}. An easy computation shows that for $\ell = 16$, $\lambda = 86$

gives us the desired error probability, and it follows that $\lambda = 87$ works for $\ell = 32$.

We have chosen the parameter values as described above for our implementation, but it is also possible to argue for different choices. One could argue, for instance, that breaking our scheme should be as hard as factoring the (1024 bits) modulus using the best known algorithm, even when the generic attack is used. Based on this, t should probably be around 200. One could also argue that having one comparison fail is not as devastating as having the cryptosystem broken, so that one could perhaps live with a smaller value of λ than what we chose. Fischlin mentions an error probability of 2^{-40} as being acceptable. These questions are very subjective, but fortunately, the complexities of the protocols are linear in t and λ, so it is easy to predict how modified values would affect the performance data we give below. Since we find that our protocol is about 10 times faster, it remains competitive even with $t = 200$, $\lambda = 40$.

6.2 *Implementation*

To evaluate the performance of our proposed protocol we implemented it along with the modified version of the protocol by Feschlin (2001) described above. The implementation was done in Java 1.5 using the standard BigInteger class for the algebraic calculations and Socket and ServerSocket classes for TCP communication. The result is two sets of programs, each containing a server, an assisting server, and a client. Both implementations weigh in at about 1300 lines of code. We have naturally tried our best to give equal attention to optimisations in the two implementations.

We tested the implementations using keys of different sizes (k in the range of 512–2048 bits) and different parameters for the plaintext space ($\ell = 16$ and $\ell = 32$). We fixed the security parameters to $t = 160$ and $\lambda = 86$ which, as noted above, should give a comparable level of security.

The tests were conducted on six otherwise idle machines, each equipped with two 3 GHz Intel Xeon CPUs and 1 GiB of RAM. The machines were connected by a high-speed LAN. In a real application the parties would not be located on the same LAN: for credibility the server and assisting server would have to be placed in different locations and under the control of different organisations (e.g. the auction house and the accountant), and the client would connect via a typical internet connection with a limited upstream bandwidth. Since the client is only involved in the initial sharing of his input, this should not pose a big problem – the majority of network traffic and computations are done between the server and assisting server, who, presumably, have better internet connections and considerable computing power.

The time complexity is linear in ℓ, so using 16 bits numbers instead of 32 bits numbers cuts the times in half. In many scenarios one will find 16 bits to be enough, considering that most auctions have a minimum required increment for each bid, meaning that the entire range is never used. As an example, eBay require a minimum increment which grows with the size of the maximum bid meaning that there can only be about 450 different bids on items selling for less than $5000 (eBay Inc., 2006). The eBay system solves ties by extra small increments, but even when one accounts for them one sees that the 65,536 different prices offered by a 16 bits integer would be enough for the vast majority of cases.

6.3 *Benchmark results*

The results of the benchmarks can be found in Table 1 with a graph in Figure 3. From the table and the graph it is clear to see that our protocol has performed significantly faster in the tests than the modified Fischlin protocol. The results also substantiate our claim that the time taken by an operation is proportional to the size of ℓ and that we do indeed roughly halve the time taken by reducing the size of ℓ from 32 to 16 bits.

Table 1 Benchmark results

k	DGK_{16}	F_{16}	DGK_{32}	F_{32}
512	82	844	193	1743
768	168	1563	331	3113
1024	280	2535	544	5032
1536	564	4978	1134	10135
2048	969	8238	1977	16500

Note: The first column denotes the key size k, the following columns have the average time to a comparison. The average was taken over 500 rounds, after an initial warm-up phase of 10 rounds. All times are in milliseconds. The abbreviation 'DGK' refers to our protocol and 'F' refers to the modified Fischlin protocol. The subscripts refer to the ℓ parameter used in the timings.

We should note that these results are from a fairly straight-forward implementation of both protocols. Further optimisations can likely be found, in both protocols.

7 Conclusion

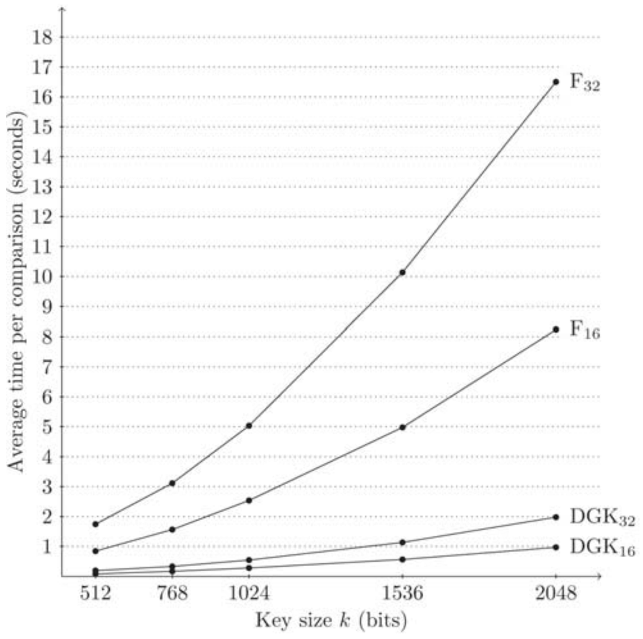
This paper has demonstrated a new protocol for comparing a public and a secret integer using only two parties, which among other things has applications in online auctions. Our benchmarks suggest that our new protocol is highly competitive and reaches an acceptably low time per comparison for real-world application.

We have also shown how to extend the protocol to the more general case where we have two secret integers and to the active security case. However, further work is needed to evaluate the competitiveness of the extended protocols.

Acknowledgements

The authors would like to thank Tomas Toft, Rune Thorbek and Thomas Mølhave for their comments and suggestions.

Figure 3 Graph of the data from Table 1 on the preceding page



References

Blake, I.F. and Kolesnikov, V. (2004) 'Strong conditional oblivious transfer and computing on intervals', in P.J. Lee, (Ed). *ASIACRYPT'04*, Volume 3329 of *Lecture Notes in Computer Science*, Springer, pp.515–529.

Blake, I.F. and Kolesnikov, V. (2006) 'Conditional encrypted mapping and comparing encrypted numbers', in G. Di Crescenzo and A. Rubin, (Eds). *FC 06*, Volume 4107 of *Lecture Notes in Computer Science*, Springer.

Brand, S. (1997) 'Rapid demonstration of linear relations connected by boolean operators', in J. Stern (Ed). *EUROCRYPT'97*, Volume 1233 of *Lecture Notes in Computer Science*, Springer, pp.318–333.

Canetti, R. (2000) 'Security and composition of multiparty cryptographic protocols', *Journal of Cryptology*, Vol. 13, No. 1, pp.143–202.

Cramer, R., Damgård, I. and Nielsen, J.B. (2001) 'Multiparty computation from threshold homomorphic encryption', in B. Pfitzmann (Ed). *EUROCRYPT'01*, Volume 2045 of *Lecture Notes in Computer Science*, Springer, pp.280–299.

eBay Inc. (2006) 'Bid increments', Available at: http: //pages.ebay.com/help/buy/bid-increments.html

Fischlin, M. (2001) 'A cost-effective pay-per-multiplication comparison method for millionaires', in D. Naccache (Ed). *CT-RSA'01*, Volume 2020 of *Lecture Notes in Computer Science*, Springer, pp.457–472.

Fujisaki, E. and Okamoto, T. (1997) 'Statistical zero knowledge protocols to prove modular polynomial relations', in B.S.K. Jr. (Ed). *CRYPTO'97*,volume 1294 of *Lecture Notes in Computer Science*,Springer, pp.16–30.

Garay, J., Schoenmakers, B. and Villegas, J. (2007) 'Practical and secure solutions for integer comparison', in T. Okamoto and X. Wang (Ed). *PKC'03*, Volume 4450 of *Lecture Notes in Computer Science*, Springer, pp.330–342.

Groth, J.(2003) 'A verifiable secret shuffle of homomorphic encryptions', in Y. Desmedt (Ed). *PKC'03*, Volume 2567 of *Lecture Notes in Computer Science*, Springer, pp.145–160.

Groth (2005) 'Cryptography in subgroups of $\mathbb{Z}_n$', in J. Kilian (Ed). *TCC'05*, Volume 3378 of *Lecture Notes in Computer Science*, Springer, pp.50–65.

Naor, M., Pinkas, B. and Sumner, R. (1999) 'Privacy preserving auctions and mechanism design', *EC'99*, New York: ACM Press, pp.129–139.

Paillier, P. (1999) 'Public-key cryptosystems based on composite degree residuosity classes', in J. Stern (Ed). *EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, Springer, pp.223–238.

Pohlig, S.C. and Hellman, M.E. (1978) 'An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance', *IEEE Transactions on Information Theory*, Vol. 24, pp.106–110.

Rode, M. (2004) 'Debat: reglerne er strammet op', *Morgenavisen Jyllands-Posten*, Available at: http://www.jp.dk/morgenavisen/ mmeninger:aid=2690052/.

Schnorr, C-P. (1991) 'Efficient signature generation by smart cards', *Journal of Cryptology*, Vol. 4, No. 3, pp.161–174.

Toft, T. (2007) 'Primitives and applications for multi-party computation', PhD thesis, University of Aarhus, Aarhus, Denmark.

Notes

[1]In Blake and Kolesnikov (2004, 2006) the emphasis is on using the comparison to transfer a piece of data, conditioned on the result of the comparison. For this application, their solution has advantages over Fischlin's, even though the comparison itself is slower.

[2]We need to send $\ell \log \ell$ bits, and public-key systems typically have $\theta(k)$-bit plaintexts and ciphertexts.