An Efficient Off-line Electronic Cash System Based On The Representation Problem

Stefan Brands

CWI
P.O. Box 4079, 1009 AB Amsterdam
The Netherlands
e-mail: brands@cwi.nl

Abstract

We present a new off-line electronic cash system based on a problem, called the representation problem, of which little use has been made in literature thus far. Our system is the first to be based entirely on discrete logarithms. Using the representation problem as a basic concept, some techniques are introduced that enable us to construct protocols for withdrawal and payment that do not use the cut and choose methodology of earlier systems. As a consequence, our cash system is much more efficient in both computation and communication complexity than previously proposed systems.

Another important aspect of our system concerns its provability. Contrary to previously proposed systems, its correctness can be mathematically proven to a very great extent. Specifically, if we make one plausible assumption concerning a single hash-function, the ability to break the system seems to imply that one can break the Diffie-Hellman problem.

Our system offers a number of extensions that are hard to achieve in previously known systems. In our opinion the most interesting of these is that the entire cash system (including all the extensions) can be incorporated straightforwardly in a setting based on wallets with observers, which has the important advantage that double-spending can be prevented in the first place, rather than detecting the identity of a double-spender after the fact. In particular, it can be incorporated even under the most stringent requirements conceivable about the privacy of the user, which seems to be impossible to do with previously proposed systems. Another benefit of our system is that framing attempts by a bank have negligible probability of success (independent of computing power) by a simple mechanism from within the system, which is something that previous solutions lack entirely. Furthermore, the basic cash system can be extended to checks, multi-show cash and divisibility, while retaining its computational efficiency.

Although in this paper we only make use of the representation problem in groups of prime order, similar intractable problems hold in RSA-groups (with computational equivalence to factoring and computing RSA-roots). We discuss how one can use these problems to construct an efficient cash system with security related to factoring or computation of RSA-roots, in an analogous way to the discrete log based system.

Finally, we discuss a decision problem (the decision variant of the Diffie-Hellman problem) that is strongly related to undeniable signatures, which to our knowledge has never been stated in literature and of which we do not know whether it is in $\mathcal{BPP}$. A proof of its status would be of interest to discrete log based cryptography in general. Using the representation problem, we show in the appendix how to batch the confirmation protocol of undeniable signatures such that polynomially many undeniable signatures can be verified in four moves.

AMS Subject Classification (1991): 94A60
CR Subject Classification (1991): D.4.6
Keywords and Phrases: Cryptography, Electronic Cash, Representation Problem

1 Introduction

Over the past years, quite some cryptographic research effort has been put in the design of off-line electronic cash systems that can not only guarantee security for the bank (and shops), but also absolute privacy for the users. As in many areas of cryptography, it is of interest to have alternatives based on various underlying assumptions instead of placing all bets on one horse. The system we present here is the first to be based on a problem (the representation problem) equivalent in computational difficulty to computing discrete logarithms rather than RSA-roots. The ability to factor does not affect the security of our system.

The main design goals for our cash system were to construct a system which is much more efficient than previously proposed solutions, has a high degree of provability and extendibility, and can be incorporated into a setting based on so-called wallets with observers under even the most stringent of privacy requirements. We achieve these goals by representing an electronic coin (or check) as a commitment on a bunch of numbers that is unconditionally secure for the committer. During its life-cycle, partial information about the numbers committed to is gradually released in the form of points and lines (polynomials in general). For this commitment, we use the so-called representation problem in groups of prime order. Similar problems hold in in RSA-groups, and almost all the tools and protocols we develop for groups of prime order can be straightforwardly adapted to hold for RSA-groups, except for the withdrawal protocol.

We feel it makes no sense to discuss the features of our cash system before we have given an overview of the standard cryptographic model for off-line electronic cash systems that can guarantee anonymity (introduced by [8]), together with a discussion of what has, and has not, been achieved in this field. We therefore in Section 2 first give a general, rather detailed discussion of off-line electronic cash systems. In Section 3, we sketch the contributions of this paper, especially focusing on provability and efficiency.

Before developing the tools for our system, we discuss the environment we work in, and the basic assumptions underlying our system in Section 4. We then come, in Section 5, to the representation problem, which is at the hart of our cash system. Along with some other fundamental properties, we prove the computational equivalence of the representation problem in groups of prime order to that of computing discrete logarithms in such groups. In Section 6 we discuss the known vector addition chain techniques to efficiently compute with the basic structure of the representation problem, as well as a simple new algorithm. The applicability of such techniques is an important reason why extensions of our system (such as checks) remain computationally efficient. In Section 7, we have listed the scarce cryptographic literature we are aware of that in some way or another makes use of the representation problem. Then, in Sections 8, 9, and 10, we discuss the basic tools we use for our cash system, and derive various propositions. These include proving knowledge of a representation, a so-called restrictive blind signature protocol, and ways to release partial knowledge of a representation.

This prepares us for Section 11, in which we explain our basic cash system and show specific protocols for withdrawal, payment and deposit. Following this is a discussion of the correctness (privacy and security) of our basic cash system. Although we cannot completely prove the correctness in a mathematically rigorous way (for reasons discussed in Section 3), the proof in our opinion comes very close. In Section 13, the extension of our system to one that includes electronic checks is discussed. The extension to divisibility is outlined in Section 14. Since transferability can be achieved in our system using the standard technique described in [1, 14], we skip a description of this extension. In Section 15 we analyze the efficiency of our system (including the extensions), and compare it to the efficiency of other systems appearing in literature.

Then, in Section 16, we discuss how to incorporate our entire cash system straightforwardly into a setting in which users have wallets with embedded observers (instead of user-modules only), even under the most stringent of requirements concerning privacy. This is followed in section 17 by a discussion of how to use several variations of the representation problem in RSA-groups for off-line electronic cash systems (including checks and wallets with observers). Almost all the tools and protocols we use for our discrete log based cash system can be adapted fairly easy to hold for the RSA representation problems, except for (unfortunately) the withdrawal protocol. We end this paper sketching some possible future directions and some open problems in Section 18. This section includes a discussion of a decision problem that to the best of our knowledge has never been stated in literature. We believe that a proof of its status is of interest not only to digital signature schemes (and hence blind signature schemes) based on discrete logarithms, but also to discrete log based cryptography in general.

2 Off-line electronic cash systems

It is often believed that electronic cash systems cannot simultaneously offer privacy for the users as well as security for the bank(s) (and shops). Many of the systems that are nowadays in use completely lack anonymity of users, and in addition are on-line in order for shops to be able to check the credibility of payers. With the advent of public-key cryptography, techniques have been developed that show that this belief is unjustified. These techniques, initialized by [8], allow the construction of off-line (!) electronic cash systems that are secure (albeit under certain intractability assumptions) for the bank, yet at the same time honest users of the system are guaranteed to remain completely anonymous. This holds in a very strong sense: the security of banks is not compromised even if all users and shops collaborate in such an attempt, and the privacy of honest users cannot be violated in any cryptanalytic way even under adversarial behaviour of the bank in coalition with all the shops. The fact that such systems can be off-line reduces a lot of the overhead, expenses and inflexibility of on-line cash systems.

2.1 The cryptographic model for off-line electronic cash

A model for off-line electronic cash that encompasses both the security and the privacy was introduced in [8], and a system was proposed that seems to fit the model (although the nature of the system is such that very little can be proven about this). Since then quite some other systems have been proposed using basically the same ideas (see [1, 7, 8, 22, 23, 29, 30, 35]). In that sense, our system is no exception.

In the model of [8], there are three distinct types of participant: a set of users $\{\mathcal{U}_1, \ldots, \mathcal{U}_k\}$, a set of shops $\{\mathcal{S}_1, \ldots, \mathcal{S}_l\}$ and a bank $\mathcal{B}$. For reasons related to mathematical rigor, k and l have to be polynomial in the length of the security parameters of the system. Throughout the system, users, shops and the bank correspond to (probabilistic) algorithms with polynomial-time computing power. In a realistic implementation of such a system, users will be represented by a user-module (which can be thought of as a smart-card, or perhaps even a personal computer or workstation), and so one should think of $\mathcal{U}$ being the user-module. The user-module is also called a wallet since it actually carries money, albeit in digital form rather than physical. Likewise, $\mathcal{B}$ and each shop $\mathcal{S}$ in a realistic situation would actually be represented by some (more powerful) computing devices.

When a user $\mathcal{U}$ in the withdrawal phase contacts $\mathcal{B}$ to obtain information that is worth some fixed amount of money (called a coin), the latter subtracts the appropriate amount from $\mathcal{U}$'s account at this very moment. When $\mathcal{U}$ wishes to pay later at a shop $\mathcal{S}$ (payment phase), he reveals the information to $\mathcal{S}$, which verifies that this information is indeed worth the appropriate amount of money. Since the system is off-line, only after some some period of time (e.g. at the end of the week) does $\mathcal{S}$ send all the information it gathered to $\mathcal{B}$ (deposit phase). $\mathcal{B}$ also verifies that the information is valid and, if so, deposits the appropriate amount of money on the account of $\mathcal{S}$.

The privacy requirement for the users is that payments made by users should not be linkable (informally, linkability means that the a posteriori probability of matching is nonnegligibly greater than the a priori probability) to withdrawals, even when banks cooperate with all the shops (untraceability). Untraceability guarantees that users remain anonymous, since their identity is only linked to withdrawals. For this, it is necessary and sufficient that the information that $\mathcal{U}$ in the payment protocol reveals to $\mathcal{S}$ is statistically independent of the information that he gets from $\mathcal{B}$ during the withdrawal protocol. Since the withdrawn information in some way has to be digitally signed with a secret key of $\mathcal{B}$, at the withdrawal phase a suitable blind signature protocol (see [11]) must be used. Usually the bank sets the security parameters, and users have no influence over this, so it is customary in cryptographic literature to assume that the bank and the shops have unbounded computing power when trying to compromise privacy of the users. Often an even stronger requirement than untraceability is made, called unlinkability. Unlinkability means that it is impossible for the bank (again colluding with the shops and having infinite computing power) to link at least two payments made by the same user. The ability to link payments that belong to the same user also is a threat to the privacy of the user (although not a purely cryptographic one, since the entire chain of payments is still unlinkable to a withdrawal by the untraceability property), since if the user would be identified in one payment (e.g., because the shopkeeper knows him), all the payments in the chain are recognized as having been made by him (so the only situation in which this really does not matter is when all these payments are made in the same transaction). Note that the privacy requirement implies that wallets should be freely obtainable anywhere, in fact users should be allowed to make their wallets themselves, since wallets that are not under control of the user can leak information concerning the user's identity at payment time.

The security for the bank consists of users not being able to forge cash, and there must be some way to prevent users from spending the same cash more than once (double-spending). Double-spending is clearly a major concern in off-line electronic cash systems with user-controlled wallets, since digital information is easy to copy. Contrary to on-line systems, where the deposit is immediately performed at the time of payment, in an off-line system there is no way to prevent double-spending at the time of payment by purely cryptographic means. It seems that one must necessarily resort to tamper-resistant wallets, thus getting in conflict with the privacy requirement. However, one can resolve this problem by using a cryptographic technique that allows the bank to catch double-spenders after the fact. This technique consists of encoding the withdrawer's identity in some way in the information he obtains during the withdrawal phase. In the payment phase, the user then has to reveal some partial information (depending on a challenge of the shop) such that when double-spending is detected at deposit phase, the identity of the user can be computed efficiently from the two (different) pieces of partial information. However, by the privacy requirements, when the users are honest their privacy should be unconditional, i.e. the identity should not be extractable from one such piece. Observe that this implies that the payment protocol necessarily has to be of the challenge-response type. Furthermore, from a practical point of view it implies that this model for off-line cash is not suitable for high-value payments, because a double-spender can for example leave the country after double-spending (and before deposit). High-value payments would therefore typically be made on-line.

The technique to catch double-spenders after the fact introduces another way for banks to compromise users, which is unrelated with their privacy. Namely, the bank can try to falsely accuse a user of having double-spent the same information. This is called a framing attempt of the bank. Therefore, there must also be a way for honest users to prove their innocence before a judge.

If one can come up with an efficient system that provably satisfies all the requirements of this model, it seems that one would have an almost ideal off-line cash system. However, there is one drawback. The term double-spending actually refers to spending the same piece of information more than once (not merely twice). Even if the system is used for low-value payments only (say up to 50 dollars), the fact that double-spending will be detected only afterwards implies that a user can spend the same $50 worth of information many times and still be able to illegitimately make a large profit (say, the same as could be achieved in a high-value payment when spending the same information twice) before being identified.

2.2 Electronic cash systems using wallets with observers

Recently, in [15], a new kind of transaction setting was proposed which, when used for off-line electronic cash purposes, can prevent double-spending, rather than detect it after the fact. That is, this setting can offer all the benefits concerning privacy and security we discussed, and yet does not suffer from the double-spending problem. It seems intuitively clear that this setting therefore cannot be based on user-controlled wallets only, and this is true indeed. The new setting uses wallets that have so-called observers embedded within them. An observer is a small tamper-resistant device that represents the organizations in the system (the bank, in the off-line cash model). By way of convention, we will call the wallet the ensemble of user-module and observer.

The idea of having an observer is that it can be incorporated in the wallet in such a way that no user-module can do a transaction on its own: in order for any transaction protocol to be executed by the wallet, it needs help (secret information) from the observer. In effect, the observer authorizes the transaction. For the double-spending problem this would simply mean that the observer takes part in the payment protocol, and would register information that is used in a payment. When a user wants to spend the same information a second time, the observer simply does not authorize the transaction. In order to be able to double-spend, users therefore are faced with the problem of having to break the tamper-resistance of the observer: the user can then find out the secret information of the observer, and have the user-module simulate the part of the observer in the payment protocol. A secure off-line cash system must therefore definitely be such that even in that case the same security as achievable in the setting with user-modules only is guaranteed, i.e., the double-spender is identified after the fact. Not minding about additional properties for the moment, an efficient system which provably accomplishes all this in our opinion can be qualified as being close to the ideal off-line electronic cash system.

So far, so good. However, it seems that we have actually arrived at an impasse: how is the privacy of the users to be guaranteed if they carry a tamper-resistant module with them which takes part in all the protocols, and has to inform e.g. the shop whether a certain action of the user is legitimate? It seems that the observer can always be programmed (by an adversarial provider) such that it can leak information related to the identity of its accompanying user-module, and hence the person to whom the wallet belongs. If users would recognize such an attempt, it would not be much of a problem since they would complain about it (and the system would be stopped). However, any unnoticeable information that flows from the observer to e.g. the shop (or vice versa) that is not specified by the protocol can greatly compromise the user's privacy, in fact to such an extent that there is no benefit over using a system with tamper-resistant user-modules (i.e., no privacy guarantee) in the first place. An example of such information is a random challenge that the shop sends to the wallet: if the observer gets to know it, the shop can encode compromising information in it (using an encoding recognizable by the observers) without the user being aware of it (inflow). Vice versa, the observer could try to secretely encode the identity of its owner into the information sent to e.g. the shop (outflow), which usually will be an even more serious threat to the privacy of the user.

It may be somewhat of a revelation that there is no impasse at all (see [15]). As a necessary (but not sufficient) requirement, the observer must thereto obviously be incorporated in the user-module in such a way that any message it sends to the outside world has to pass through the user-module, thereby enabling the user-module to verify (and moderate) these messages. In this way, the user-module can recognize attempts of the observer to leak information (outflow), and vice versa. However, it should not be able to moderate to such an extent that it can authorize transactions by itself. Using special cryptographic protocols, all the requirements concerning security can be satisfied, as well as those related to privacy. In particular, protocols can be constructed such that there is no (undetectable by the user-module) inflow or outflow. This implies for example that all random numbers chosen in the protocol by the observer or the outside party (i.c. the shop) are moderated by the user-module before they are sent through.

In [17] the privacy aspect of the wallet setting has been investigated under the most stringent requirement one can think of: even if each observer were to store all information it receives during the period it is embedded within a user module, it still should be impossible (independent of computing resources) to match a user module and an observer afterwards by looking at the information inside the observers and all information gathered by the organizations (this could be called traceability after the fact, since once e.g. the bank can do so, all the payments made with the wallet can be traced). This possibility is not excluded by preventing inflow and inflow, since for example a single random number known to both an observer and a shop would enable linking: the fact that the user-module took part in generating it (so that no information could be encoded within it, thus preventing both inflow and outflow) is irrelevant for this. Such mutually known information which enables linking is called shared information, and it comprises both inflow and outflow. One of the essential techniques needed to prevent shared information is that of divertability of protocols (see [31]). For further information about the cryptographic techniques used to prevent inflow, outflow and, if desirable, shared information while retaining security, we refer the reader to [3, 4, 12, 15, 17].

We note that prevention of shared information may in reality not always be that important an issue, since, in contrast to inflow and outflow, shared information cannot affect the privacy of the user during the time it is with the user. It is only when observers must systematically be handed in to the provider for some reason, that information about the payment history of their owners can be extracted if there is shared information. In an electronic cash system this is quite realistic to assume, and therefore in our opinion it is of interest to have electronic cash systems using wallets with observers that satisfy even the requirement of no shared information. In any case, one can consider the ease with which an off-line electronic cash system can be incorporated into the wallet setting with observers under the condition of no shared information to be a testcase for the flexibility of the cash system under consideration.

2.3 Towards provably correct and efficient systems

Summarizing the discussion in the previous subsections, it seems that cryptographic techniques are available that enable the construction of off-line electronic cash systems that can guarantee privacy and security at the same time. If one is content with detecting double-spending after the fact, it can be realized using wallets which are totally under control of the users (i.e. no observers). When double-spending is to be prevented in the first place, it can be realized in a setting in which users have wallets with embedded observers.

However, the first mathematical proof of correctness of the existing cash systems has yet to be given. In fact, all cash systems in literature have in common that they use such complicated constructions that almost nothing has been proven about their correctness, and intuitive arguments are given instead. Moreover, the entire discussion so far has not been concerned with efficiency considerations at all. Therefore, from a practical point of view, it remains to construct a system which is really efficient, and of which we can mathematically prove that all the requirements are met (correctness). With hardware technology having rapidly progressed over the past decade, systems that would be efficient from a complexity-theoretic viewpoint (i.e., small degrees of polynomials in computation-time estimates, protocols with only a few moves and consisting of a few numbers only) could be implemented with current hardware (e.g., in smart-cards) in a way that satisfies all the speed and storage requirements usually imposed.

Concerning the mathematical provability of correctness, one can argue that this is perhaps the most important issue to be resolved. Unfortunately, the issue of $\mathcal{P}$ versus $\mathcal{NP}$ is likely to remain unresolved for a long time to go, and hence the best one can actually do is to reduce certain well-known intractable problems (such as factoring and extracting discrete logarithms) to the security of the system. However, the current state of cryptographic proof techniques is that no mathematical framework is known to derive such a reduction in for complex cryptographic systems (i.e. systems consisting of more than one protocol, like off-line electronic cash systems).

2.4 Additional properties

Additional properties that one may impose on an ideal off-line electronic cash system are transferability and divisibility of electronic cash, and electronic checks. Transferability means that both shops and users can act as payers and payees, and can pay with money they receive to other participants in the system without intervention of the bank. A generic method to add transferability to essentially any off-line electronic cash system (including ours) can be found in [1, 14]. There are some drawbacks of transferability, the most important of them being that information-theoretically it can be proven that, during transfer, electronic cash grows in size (see [14]).

Divisibility (as achieved in [30]) means that electronic cash can be split into pieces in an arbitrary way, summing up to the total amount of the withdrawn information, such that each piece can be spent individually. This also seems to have various drawbacks related to privacy and especially efficiency, although in contrast to transferability no information-theoretic results like that of [14] are known. A particular problem with divisibility is that it seems very difficult to achieve without introducing (full) linkability of the payments.

Electronic checks can be spent for any amount up to a certain maximum, thereby reducing the deposit workload since payers need not pay a certain amount with several coins (each of which during deposit phase is accompanied by a challenge and response), but can pay with a single check. During withdrawal of the check, the user is charged for the maximum amount, and he can get a refund of the unspent part later on. For this, an extra protocol (refund protocol) is needed in the model.

Using user-controlled modules only, each of these three additional properties has its own drawbacks. However, in many practical situations there is no need at all for transferability, and in fact it would probably even be rather awkward since transferred money grows. The same holds to some extent for divisibility, unless it can be achieved efficiently without linkability. In contrast, one may really want to use checks in a system, which in our opinion is the only additional property which has a fair chance of being efficiently realizable without the drawbacks. In a system with a really efficient payment protocol (from the viewpoint of storage and amount of computation needed to verify the validity of a coin) there is probably no advantage at all in having electronic checks, since the deposit phase will not be that less inefficient, whereas on the positive side there is no need for a refund protocol. If the payment is not that efficient, it seems best to have a cash system in which one can pay with checks as well as with coins. Paying certain amounts with a check can then be more efficient if the amount to be paid would require many coins, and vice versa it makes no sense to pay with a check an amount of money that can be paid with just a few coins.

Contrary to the setting with user-controlled modules only, in the setting based on wallets with observers it is conceivable that all the additional properties can be realized efficiently without most of the drawbacks, since one can have the observer handle certain problems which are hard to solve in the setting with user-modules only. Basically, there are two possible ways to go in achieving the additional (but also the basic) properties in the observer setting. The first is to transplant a (seemingly) correct system designed for a setting without observers, into one with observers. If one is not concerned about prevention of shared information, this can often be done straightforwardly. In case shared information must also be prevented, this becomes a much more difficult thing to do, since in essence it depends on whether the protocols of the system that is to be transplanted can be diverted. The other way is to make use of unique possibilities of the three-party wallet-with-observer setting. For example, although we can transplant our check system over into the observer setting, checks in this setting can probably be achieved more efficiently by having the observer take care of the amount the user should get refunded. At the point of writing of this paper, the second way to deal with off-line cash in wallets with observers is almost unexploited.

3 Features of our cash system

In this section we sketch the contribution of this paper to off-line electronic cash systems. With the discussion of the previous section in mind, a categorization can be made into several categories:

(Efficiency) Our system is much more efficient in communication complexity (both in the withdrawal and the payment phase) and computation complexity than previously proposed solutions. In particular, the withdrawal and payment protocols both consist of only three moves, and the deposit protocol requires one move. Using the standard technique of having the challenge of the shop be a one-way function of certain information, we can condense the payment protocol to a single move. All previously proposed solutions use binary challenge-response protocols in the withdrawal phase (to ensure the user's identity will be in the signed information he receives) and in the payment phase (to ensure that he will be caught when double-spending), and therefore need polynomially many repetitions (often done in parallel) to achieve the desired degree of security. In our check extension, the withdrawal protocol consists of four moves, the payment and refund of three moves, and the deposit of one. For the payment and the refund, the same remarks hold concerning the challenge as above. Since vector addition chain techniques can be applied, even the verification relations are much more efficient to compute than those of previous systems.

(Provability) We can prove the correctness of our cash system to a very high degree, which in our view is the main problem with earlier systems ([1, 7, 8, 22, 23, 29, 30, 35]). Specifically, the security of our system seems to be equivalent to some well-known hard problem (the Diffie-Hellman assumption), under only a single assumption concerning an unspecified hash-function that is likely to be weaker than the Diffie-Hellman assumption.

(Extendibility) Our basic cash system can be extended in a number of ways quite easily, while retaining its efficiency. For example, users can be allowed to spend their coins k times (k-show signatures) instead of only once (one-show signatures), in such a way that $k+1$ spendings reveal the identity of the user, whereas up to k spendings still hide it unconditionally. Although all these k payments are linkable, the entire chain of k payments is still unconditionally unlinkable to a specific user. This extension does not affect the number of rounds of the protocols, and causes only a linear increase in the length of the transmitted information and (due to vector addition chain techniques) computation complexity.

Another way to extend our system is the use of checks. We construct a check system that is quite efficient, and of which our basic system with coins only is in fact just a special case. Checks with different maximum values can be used.

In a similar approach, the divisibility property can be achieved in our system. Except for the system of [30], all systems in literature lack this extension. Like with the coins, it can even be arranged that the user can spend the check in this way multiple times, without his identity getting known. We remark that all k-show extensions as well as the extension to divisible cash have the drawback of full linkability of payments, and therefore maybe are only useful in environments where users can easily make decisions with respect to how their own privacy is to be protected: in case a user then is worried about linkability of certain transactions, he can decide to spend other information in certain transactions than e.g. the unspent part of a check which has been paid with before.

Since transferability can be added to our cash system with the standard technique described in [1, 14], it therefore meets all the requirements of [30]. Moreover, there is another extension that in our opinion is quite of interest, which is that our system (including all the extensions) can straightforwardly be incorporated into a setting based on wallets with observers (see e.g. [12, 15]), thereby increasing security even more: in order to double-spend, users in addition to the cryptographic assumptions also have to break a tamper-resistant device (i.e. we have security in a very strong sense). Due to the representation problem, most of our protocols can easily be diverted (see [31]). As a consequence, the entire cash system can be incorporated in the wallet setting fairly straightforward even under the strong privacy requirement that no shared information arise. In contrast, previously proposed electronic cash systems are very hard to incorporate into the wallet setting under this requirement.

(Functionality) Our system offers some extra functionality. For example, framing attempts of the bank (given unlimited computing power throughout the entire system when trying to compromise users) are prevented by a mechanism from within the system: users will be able to prove that they are falsely being accused of e.g. double-spending. In previous systems, the identifying information of users is always completely known to the bank, hence in order to prevent framing these systems have to apply some less elegant techniques. In our system, the identifying information of users consists of a secret part which cannot be computed by the bank, regardless of computing power.

(Analogues in other groups) Almost all previously proposed cash systems make use of special tricks that are hard to adapt to other groups. In contrast, our system uses many tools which can easily be adapted to hold for RSA-groups (i.e. multiplicative groups modulo the products of two primes) as well, due to the representation problem. More specifically, there are a few variations with computational difficulty equivalent to factoring or computing RSA-roots (see Section 17). In fact, the only tool that we do not know how to adapt for RSA-groups is the blind signature scheme, which is needed for the withdrawal protocol. We state this as an open problem in Section 18.

Despite these nice features, there are still several parts of our system that could be improved. It may well be that a more efficient restrictive blind signature scheme exists from the point of view of storage space, since in our cash system one signature of the bank consists of four numbers. In Section 18 we discuss one of the problems in coming up with a more efficient signature scheme. The blind signature scheme is also the reason that the computational effort needed to break the system is no greater than that needed to break the Diffie-Hellman problem. It would of course be nicer to have the security equivalent to the Discrete Log problem.

3.1 A brief discussion of the techniques that we use in our cash system

The electronic cash systems that have been proposed in literature thus far share the fact that, presuming that they are indeed secure, it is hard to prove anything about this (i.e., come up with reductions). With this, we are not aiming at a rigorous mathematical proof of correctness, since, as we earlier remarked, as yet a mathematical framework for this is not available. Instead, we mean the kind of partial proofs of security that have become commonplace in cryptography, and which go a long way towards such a rigorous proof. One of the main problems in coming up with formal proofs of security of complex cryptographic systems, such as off-line electronic cash systems, is that it seems hard to prove that the composition of minimum-knowledge protocols is minimum-knowledge. Although such a general result is known for (auxiliary input) zero-knowledge protocols, signature schemes can by definition not be derived from zero-knowledge protocols. That is, the requirements impose that we cannot resort to these composition theorems when proving correctness of electronic cash systems.

Currently, the best one can do is to try to get as close as possible to a proof of correctness. However, with the previously proposed systems, even this turns out to be very difficult. Although this is often thought to be an inherent aspect of designing electronic cash systems (as well as of other complex models), there are some explicit causes that can be pinpointed. The most obvious of these is the use of several unspecified functions in the protocols, of which the interactions are almost impossible to analyze. Typically, such functions are introduced because manipulations in various groups throughout the system require mappings between the elements of the various groups. Another reason for their occurrence it that certain requirements are often hard to solve (constructing off-line electronic systems is not easy!) without introducing unspecified functions (which must then be assumed to have certain desired properties). As a result, the solution arrived at usually has a structure that is hard to subject to analysis.

Hence, it is crucial to start with a flexible underlying problem from which the system is designed. In that way, one is much more likely not to have to use unspecified functions. For this reason, our electronic cash system makes use of essentially only one basic underlying problem, the representation problem, which allows very flexible manipulations. A system with protocols that consist only of a few moves clearly also helps to prove correctness.

Being able to detect double-spending by encoding the identity in the electronic cash is probably the single most difficult problem to resolve in privacy-protecting off-line cash systems not relying on tamper-resistance, and is what accounts for much of the inefficiency and unspecified functions. In our cash system, whether the identity of a user is in the withdrawn information (coins and checks) depends entirely on his knowledge about it. That is, how much certain withdrawn information is worth, whose identity is in it and whether it can be spent depends entirely on the knowledge of the user about the information. Although it is true that the identity must necessarily be encoded in knowledge of the payer about the signed information if untraceability is to be unconditional, and this hence is also the case in all previously proposed off-line electronic cash systems, our system is the first one in which this is used as a basic concept rather than being an awkward necessity.

This way of encoding identity in conjunction with the representation problem allows us to do virtually all manipulations in our system in two (isomorphic) groups (G_q and $\mathbb{Z}_q$) without needing to introduce any unspecified functions except one hash function. More importantly, it enables us to apply some techniques in our cash system that benefit provability of correctness and efficiency to a great extent. The first of these is applied in the withdrawal protocol, and allows us to dispose of the cut-and-choose method normally used to ensure that the information (or knowledge) a user ends up with contains his identity. Rather than having the user put his own identity in blinded numbers, and convince the bank that he did so by opening all but a few (chosen by the bank), we use what we will call a restrictive blind signature scheme. This is a blind signature scheme in which the user is restricted in the kind of blinding manipulations he can do: he can establish the statistical independence needed for untraceability, yet a certain structure (in which we encode his identity) in the knowledge he has of the blinded information will always be the same as that of the unblinded information, regardless of the specific blinding manipulations chosen by the user. As a consequence, the user need only blind a single number (instead of polynomially many, of which he opens all but some). The second technique is also applied in the blind signature scheme: we effectively separate different executions of the withdrawal protocol by having each signature consist of a new random number, generated by the bank and infeasible to compute by the user. This accounts for provability of infeasibility of chosen message and multi-user attacks applied to the withdrawal protocols. The third technique is applied in the payment protocol and uses the well-known idea that the slope of a line in the plane is completely determined by two different points on the line, yet is completely undetermined if only one point on the line is known (this technique has been used in the cash system of [22]). This allows us to construct a very fast payment protocol, since there is no need for polynomially many challenges and responses.

We use several proof-techniques in proving the security aspect of our system. The most important of these, of which we make use at various places, stems from the fact that our constructions are such that the bank need not know any more secret information (although it might) than all other participants of the system, except for the secret key it uses to make the (blind) signature with. This allows us to view the entire system as one entity which, since all participants of the protocols can be polynomial-time algorithms (even if the bank does not need to be), in effect can be simulated by a probabilistic polynomial-time algorithm. In this way, we can prove that many conceivable attacks on the system are infeasible, because they would imply that the attacker can determine some relative discrete logarithm with nonnegligible probability. This in turn implies that there must exist a feasible algorithm which computes discrete logarithms with nonnegligible probability, namely that simulating the system (simulating the random choices and actions of the participants).

As we discussed, the reason that we can derive an almost similar system in RSA-groups (if it weren't for the restrictive blind signature scheme, which we do not know how to adapt to RSA-groups) is that all our tools make use only of properties characteristic to the representation problem. Although we use the restriction of this problem to groups of prime order, variations of it in RSA-groups are also computationally difficult, and hence almost all our tools can be adapted to RSA-groups.

Due to the fact that we can divert (see [31]) all the protocols, since they are based mainly on the representation problem, we can incorporate the system in the wallet-with-observer setting even under the most stringent requirements concerning privacy (as described in [17]). In addition, the efficiency of the check and divisibility extensions of our cash system clearly benefit much from the applicability of vector addition chain techniques, and this (although to a lesser extent) also holds for the basic cash system.

4 Notation and basic assumptions

Throughout the paper, all arithmetic is performed in a group G_q of (known) prime order q (so G_q is Abelian) for which polynomial-time algorithms are known to determine equality of elements, test membership, compute inverses, multiply, and to randomly select elements. There is a vast variety of groups known to satisfy these requirements (see e.g. [27]). The advantages of working in such a group are that it is hard to distinguish between elements because they are all (except the unity element) generators of the group, and manipulating with indices is very convenient because one in effect is dealing with arithmetic in a field.

Although our results are valid for any such group, for explicitness we use the subgroup G_q of $\mathbb{Z}_p^*$, with $p = kq + 1$ a prime (for some $k \in \mathbb{N}$). In a practical implementation of our system, the length of p would probably be at least 512 bits, and that of q at least 140 bits. We remark that, when working in a group of this specific form, no methods are known to compute partial information of discrete logarithms.

When we speak of polynomial-time algorithms, we implicitly allow them to be probabilistic (more specifically, of the Monte Carlo and Las Vegas type). The terms 'negligible/overwhelming probability' and 'feasible/infeasible' have the familiar complexity-theoretical meanings.

Each protocol that can be used in our cash system is described in text and, for convenience of the reader, displayed in a figure. In the figures, we also display some extra information that is strictly speaking not part of the protocol but has to be computed, looked up or transmitted beforehand. To make clear that this information is not part of the actual protocol, we separate it from the rest of the figure using vertical dots. Where verifications are needed, some extra symbols are introduced, since otherwise the verification relations are mathematically superfluous (they are always true). Extra symbols are also used in proving various propositions about the protocols, because we then usually have to assume that (at least) one participant does not transmit the information specified in the description of the protocol.

The security of our cash system is related to the Discrete Log and Diffie-Hellman assumptions, both of which we state here with respect to G_q.

Definition 1 *Finding the unique index $\log_g h \in \mathbb{Z}_q$ of $h \in G_q$ with respect to $g \in G_q \setminus \{1\}$ is the Discrete Log problem. An algorithm is said to solve the Discrete log problem if, for inputs $g \neq 1$, h generated uniformly at random, it outputs $\log_g h$ with at least nonnegligible probability of success. The Discrete Log assumption states that there is no polynomial-time algorithm which solves the Discrete Log problem with overwhelming probability of success.*

It is well known that the Discrete Log assumption is equivalent to assuming that there cannot exist even a single $g \in G_q$ and a polynomial-time algorithm such that, on input $h \in G_q$ chosen at random, the algorithm has nonnegligible probability of outputting $\log_g h$. Clearly this increases our believe in the validity of the Discrete Log assumption. This is due to the following propositions.

Proposition 2 *If there exists an algorithm which solves the Discrete Log problem, then there exists an algorithm which solves the Discrete Log problem with overwhelming probability of success.*

Proof. To compute $\log_g h$, one can feed (g^{r_1}, h^{r_2}) for random choices of r_1, r_2 into the algorithm. After an expected number of polynomially many steps, the algorithm correctly outputs $\log_{g^{r_1}}(h^{r_2})$. One knows when this occurs since one can recognize a correct answer (!), and then $\log_g h$ can be computed by multiplying by $r_1/r_2 \bmod q$. □

PROPOSITION 3 *The following two statements are equivalent:*

(a) There exists a polynomial-time algorithm $\mathrm{A}_{(\mathrm{a})}$ *which, for randomly chosen inputs* $g \neq 1$ *and* h*, outputs* $\log_g h$.

(b) There exist a $g \neq 1$*, and a polynomial-time algorithm* $\mathrm{A}_{(\mathrm{b})}$ *which, for randomly chosen input* h*, outputs* $\log_g h$.

PROOF. Proving (a) $\Rightarrow$ (b) is obvious. To prove the other implication, suppose we have inputs g_1, h and want to receive $\log_{g_1} h$. Algorithm $\mathrm{A}_{(\mathrm{a})}$ first feeds g_1 into $\mathrm{A}_{(\mathrm{b})}$ to receive $a = \log_g g_1$, then feeds h into $\mathrm{A}_{(\mathrm{b})}$ to receive $b = \log_g h$, and outputs $b/a \bmod q$. □

Due to Proposition 2, in the latter proposition algorithms $\mathrm{A}_{(\mathrm{a})}$ and $\mathrm{A}_{(\mathrm{b})}$, rather than being deterministic, can be allowed to have success nonnegligible probability.

Observe that the Discrete Log assumption as stated here is weaker than the one that is normally encountered in literature: if one can compute discrete logarithms in $\mathbb{Z}_p^*$ with respect to arbitrary elements, then one certainly can compute them in G_q, but the reverse is not necessarily true.

DEFINITION 4 *Finding the unique Diffie-Hellman key* g^{ab} *of* $g_1 (= g^a)$ *and* $g_2 (= g^b)$ *with respect to* g *is the Diffie-Hellman problem. An algorithm is said to solve the Diffie-Hellman problem to the base* g *if, for inputs* g_1 *and* g_2 *generated uniformly at random, it outputs the Diffie-Hellman key of* g_1 *and* g_2 *with respect to* g *with at least nonnegligible probability of success. The Diffie-Hellman assumption states that, for all* $g \neq 1$*, there is no such polynomial-time algorithm.*

For the Diffie-Hellman problem, no similar results are known. There is an important cause for this unsatisfactory state of affair: the Decision Diffie-Hellman problem. Since this decision problem is also intimately related to undeniable signatures, which we in turn implicitly use in the blind signature scheme (although it is unrelated to the security of our system), we postpone further discussion to Section 18.

If we were to state the Discrete Log and Diffie-Hellman assumptions with respect to other input probability distributions than the uniform one, then clearly this would affect all statements based on these assumptions. Since incorporating the allowed input distributions into our results would distract attention too much from the essentials, we from now on by definition always imply a uniform probability distribution when we choose or generate certain elements 'at random', and denote this with the symbol "$\in_{\mathcal{R}}$".

5 THE REPRESENTATION PROBLEM

In this section we introduce the main problem underlying our constructions, and prove some basic properties.

DEFINITION 5 *Let $k \geq 2$ be a constant. A generator-tuple of length k is a k-tuple $(g_1, \ldots, g_k)$ with, for all $i, j \in \{1, \ldots, k\}$, $g_i \in G_q \setminus \{1\}$ and $g_i \neq g_j$ if $i \neq j$. An index-tuple of length k is a k-tuple $(a_1, \ldots, a_k)$ with $a_i \in \mathbb{Z}_q$ for all $i \in \{1, \ldots, k\}$. For any $h \in G_q$, a representing index-tuple (also called a representation) of h with respect to a generator-tuple $(g_1, \ldots, g_k)$ is an index-tuple $(a_1, \ldots, a_k)$ such that $\prod_{i=1}^{k} g_i^{a_i} = h$.*

In fact, k can be polynomial in the length of the input without this affecting any of our results. Since we make no use of this anywhere, we do not incorporate it into the definition.

Usually, it will be clear with respect to which generator-tuple we take the representing index-tuple, and we therefore often do not mention it explicitly. If we take $h = 1$, one representation immediately springs to mind, namely $(0, \ldots, 0)$. We call this the trivial representing index-tuple. If we would take $k = 1$ in the definition, we simply have the familiar Discrete Log situation with $a \in \mathbb{Z}_q$ representing $h \in G_q$ with respect to $g \neq 1$. We deliberately exclude this situation from the definition, since there are important distinctions with $k \geq 2$.

PROPOSITION 6 *For all $h \in G_q$ and all generator-tuples of length k there are exactly q^{k-1} representing k-tuples of h.*

PROOF. Since G_q has prime order, each element $\neq 1$ is a generator of G_q. We can therefore choose the first $k-1$ elements of the representing k-tuple $(a_1, \ldots, a_k)$ at random from $\mathbb{Z}_q$. The k-th element a_k is then uniquely determined as the discrete logarithm of $h / \prod_{i=1}^{k-1} g_i^{a_i}$ with respect to g_k. □

This simple result implies that the density of representing k-tuples of h is negligible with respect to the set containing all index-tuples of length k. Therefore, any polynomial-time algorithm that applies an exhaustive search strategy to find one has negligible probability of success. The next proposition shows that there is no essentially better strategy, assuming the Discrete Log assumption.

PROPOSITION 7 *Define V to be the set of all functions of the form $q(\cdot)/r(\cdot)$, such that $q(\cdot)$ and $r(\cdot)$ are polynomials with integer domain and integer coefficients, and $q(k) > r(k) \geq 1$ for all sufficiently large k. For any functions $f_1(\cdot), f_2(\cdot), f_3(\cdot), f_4(\cdot) \in \mathrm{V}$, the following four statements are equivalent:*

(1) There exists a polynomial-time algorithm $\mathrm{A}_{(1)}$ which, on inputs a generator-tuple of length k and $h \in G_q$, outputs a representing index-tuple of h with probability of success at least $1/f_1(|p|)$ for all sufficiently large p.

(2) There exists a polynomial-time algorithm $\mathrm{A}_{(2)}$ which, on input a generator-tuple of length k, outputs a nontrivial representing index-tuple of 1 with probability of success at least $1/f_2(|p|)$ for all sufficiently large p.

(3) There exists a $h \in G_q \setminus \{1\}$ and a polynomial-time algorithm $\mathrm{A}_{(3)}$ which, on input a generator-tuple of length k, outputs a representing index-tuple of h with probability of success at least $1/f_3(|p|)$ for all sufficiently large p.

(4) There exists a polynomial-time algorithm $\mathrm{A}_{(4)}$ which solves the Discrete Log problem with probability of success at least $1/f_4(|p|)$ for all sufficiently large p.

Proof. We only need to show probabilistic polynomial-time transformations of (4) to each of (1), (2) and (3), since we can come up easily with feasible algorithms $A_{(1)}$, $A_{(2)}$ and $A_{(3)}$ if we have $A_{(4)}$. For this, we proceed as in the proof of Proposition 6 and use the fact that the probability of success of $A_{(4)}$ can be arbitrarily boosted (see Proposition 2).

Assume that we have two elements $g, h \in G_q$, $g \neq 1$, and want to compute $\log_g h$.

(1) $\Rightarrow$ (4) Algorithm $A_{(4)}$ proceeds as follows:

- **[Step 1]** Generate a k-tuple $(u_1, \ldots, u_k)$ at random, and compute the generator-tuple $(g_1, \ldots, g_k)$ according to $g_i = g^{u_i}$ for $i \in \{1, \ldots, k\}$. Feed this generator-tuple together with h into $A_{(1)}$.
- **[Step 2]** Receive an index-tuple $(a_1, \ldots, a_k)$ from $A_{(1)}$. If it is not a representation of h, go to Step 1.
- **[Step 3]** Compute (and output) $\log_g h = \sum_{i=1}^{k} a_i u_i \bmod q$.

After an expected number of $f_1(|p|)$ repetitions of Step 2, $A_{(4)}$ receives a representation of h.

(2) $\Rightarrow$ (4) We have to apply a different strategy, since $A_{(2)}$ does not compute representing index-tuples with respect to h of one's own choice. We take advantage of the fact that for randomly chosen u_i, u_j, there is no way to distinguish between h^{u_i} and g^{u_j}. Moreover, in this particular situation we exploit the fact that $g^0 = 1$ for all g. Algorithm $A_{(4)}$ proceeds as follows:

- **[Step 1]** Generate a k-tuple $(u_1, \ldots, u_k)$ at random, and compute the generator-tuple $(g_1, \ldots, g_k)$ according to $g_1 = h^{u_1}, g_2 = g^{u_2}, \ldots, g_k = g^{u_k}$.
- **[Step 2]** Generate at random a permutation $\pi(\cdot)$ of k elements. Apply $\pi(\cdot)$ to the generator-tuple computed in Step 1 and feed the resulting generator-tuple

$$(g_{\pi(1)}, \ldots, g_{\pi(k)})$$

  into $A_{(2)}$.
- **[Step 3]** Receive an index-tuple $(a_1, \ldots, a_k)$ from $A_{(2)}$. If it is not a representation of 1, or if $a_{\pi^{-1}(1)} = 0$ go to Step 1.
- **[Step 4]** Compute (and output) $\log_g h$ from the following linear equation, (induced by the equation $\prod_{i=1}^{k} g_{\pi(i)}^{a_i} = g^0$):

$$u_1 a_{\pi^{-1}(1)} \log_g h + \sum_{i=2}^{k} u_i a_{\pi^{-1}(i)} = 0 \bmod q.$$

In Step 3, in the worst case situation ($k \geq 3$), $A_{(2)}$ always sets $k-2$ elements of a representing index-tuple to zero. So, for all $k \geq 2$, with probability at least $2/k$, a representing index-tuple received in Step 3 does not have $a_{\pi^{-1}(1)}$ (the index corresponding to h) equal to zero. Since $A_{(4)}$ receives a representation of 1 in Step 3 after an expected number of $f_2(|p|)$ repetitions, in the worst case it takes an expected number of $f_2(|p|)k/2$ repetitions of Step 3 before $\log_g h$ can be extracted.

(3) $\Rightarrow$ (4) Again we have to alter the strategy somewhat, since under the Discrete Log assumption we cannot in general compute $\log_g h$. Algorithm $\mathrm{A}_{(4)}$ proceeds as follows:

(Initialization) Put $i = 1$.

[Step 1] Generate a k-tuple $(u_{i,1}, \ldots, u_{i,k})$ at random, and compute from this the generator-tuple $(g_{i,1}, \ldots, g_{i,k})$ as $g_{i,1} = h^{u_{i,1}}, g_{i,2} = g^{u_{i,2}}, \ldots, g_{i,k} = g^{u_{i,k}}$.

[Step 2] Generate at random a permutation $\pi_i(\cdot)$ of k elements. Apply the selected permutation to the generator-tuple computed in Step 1 and feed the resulting generator-tuple $(g_{i,\pi_i(1)}, \ldots, g_{i,\pi_i(k)})$ into $\mathrm{A}_{(3)}$.

[Step 3] Receive a nontrivial index-tuple $(a_{i,1}, \ldots, a_{i,k})$ from $\mathrm{A}_{(3)}$. If it is not a representation of h go to Step 1.

[Step 4] If $i = 1$, increase i by 1 and go to Step 1.

[Step 5] If $u_{2,1} a_{2,\pi_2^{-1}(1)} = u_{1,1} a_{1,\pi_1^{-1}(1)} \bmod q$, go to Step 1.

[Step 6] Compute (and output) $\log_g h$ from the following linear equation, (induced by the equation $\prod_{j=1}^k g_{2,\pi_2(j)}^{a_{2,j}} = \prod_{j=1}^k g_{1,\pi_1(j)}^{a_{1,j}}$):

$$u_{2,1} a_{2,\pi_2^{-1}(1)} \log_g h + \sum_{l=2}^{k} u_{2,l} a_{2,\pi_2^{-1}(l)} = u_{1,1} a_{1,\pi_1^{-1}(1)} \log_g h + \sum_{l=2}^{k} u_{1,l} a_{1,\pi_1^{-1}(l)} \bmod q.$$

In Step 2, if $i = 1$ we can as well take the identity permutation.

It takes an expected number of $f_3(|p|)$ repetitions of Step 3 before we receive a representation for the first time ($i = 1$). For $i = 2$ it takes a worst-case expected number of $k f_3(|p|)$ repetitions of Step 3 before we can extract $\log_g h$, since we have to repeat Step 5 a worst-case expected number of k times. Therefore even in the worst case we can extract $\log_g h$ in expected polynomial time.

Note that, for all reductions, the probability that the received index-tuple in Step 1 is not a generator-tuple is negligible even if k were polynomial in $|p|$ (despite the birthday paradox).

In the proof of the last two implications, we can optimize the resulting algorithm $\mathrm{A}_{(4)}$ considerably by inputting blinded versions of h in $\lfloor k/2 \rfloor$ elements instead of in just one. Note that algorithm $\mathrm{A}_{(4)}$ as constructed from $\mathrm{A}_{(2)}$ respectively $\mathrm{A}_{(3)}$ is of the Las Vegas type even if $\mathrm{A}_{(2)}$ and $\mathrm{A}_{(3)}$ always give correct outputs. $\square$

We now define the **representation problem** in its general form (using a standard specification format).

Name: Representation problem.

Instance: A group G, elements $g_1, \ldots, g_k \in G$ (with k polynomial in $|G|$), $h \in G$.

Question: Is there a representation of h with respect to $(g_1, \ldots, g_k)$ and, if so, find one.

In our cash system we only use a restricted form of this problem, the representation problem for groups of prime order. Proposition 7 states that the representation problem for groups of prime order is equivalent in computational difficulty to the Discrete Log problem.

We note that similar restrictions of the representation problem to other groups are very likely to have the same kind of applicability as the representation problem in groups of prime order. For example, if the input group is always an RSA-group, the problem can be shown to be equivalent in computational difficulty to factoring. One then is interested in solutions of $X_1^{a_1} \cdots X_k^{a_k} \bmod n$ in terms of the a_i's, for fixed X_i's. Another variation is to find the X_i's, and fix the a_i's, which is a problem as difficult as extracting RSA-roots. A combination of these two variants is to find solutions of mixed forms $X_1^{a_1} \cdots X_k^{a_k} B_1^{y_1} \cdots B_k^{y_k} \bmod n$ in terms of $X_1, \ldots, X_k, y_1, \ldots, y_k$ for fixed a_i's, B_i's. In Section 17, we discuss how these variations can be used to construct a cash system with the techniques of this paper.

In Proposition 7, we can allow certain elements of the input generator-tuple to be selected with a different probability distribution than the uniform one. To the extreme, the proposition still holds if we substitute (1) respectively (2) by:

(1) There exists a generator-tuple of length k and an algorithm $\mathrm{A}_{(1)}$ which, on input $h \in G_q$, outputs a representing index-tuple of h with probability of success at least $1/f_1(|p|)$ for all $|p|$ large enough.

(2) There exists a generator g_k and a polynomial-time algorithm $\mathrm{A}_{(2)}$ which, on input a generator-tuple $(g_1, \ldots, g_{k-1})$, outputs a nontrivial representing index-tuple of 1 with respect to $(g_1, \ldots, g_k)$ with probability of success at least $1/f_2(|p|)$ for all $|p|$ large enough.

Similar changes in the distribution of the input generator-tuple can be applied to most of the results in this paper, but we will not mention these from now on since they are always fairly obvious.

An immediate but very important consequence of Proposition 7 is the following:

COROLLARY 8 *Under the Discrete Log assumption, there cannot exist a polynomial-time algorithm which, on input a generator-tuple $(g_1, \ldots, g_k)$ chosen at random, outputs a number $h \in G_q$ and two different representing index-tuples of h with nonnegligible probability.*

PROOF. If there were such an algorithm $\mathrm{A}_{(1)}$, we could use it to build the following algorithm:

[Step 1] Feed the generator-tuple $(g_1, \ldots, g_k)$ into $\mathrm{A}_{(1)}$ and receive h and two representing index-tuples $(a_1, \ldots, a_k)$ and $(a_1', \ldots, a_k')$ of h.

[Step 2] Output $(a_1 - a_1' \bmod q, \ldots, a_k - a_k' \bmod q)$.

We now have constructed algorithm $\mathrm{A}_{(2)}$ of Proposition 7, and therefore have a contradiction with the Discrete Log assumption. □

A consequence of this is that, for n such that $0 \le n \le k$, under the Discrete Log assumption any polynomial-time algorithm which, on input a generator-tuple $(g_1, \ldots, g_k)$ chosen at random, outputs $h \in G_q$ and at least $1 + q^{n-1}$ different representing index-tuples of h, must have access to the relative discrete logarithms of at least n distinct pairs (g_i, g_j).

The above results imply that one can regard the function $f(\cdot)$, defined as

$$f(a_1, \ldots, a_k) = \prod_{i=1}^{k} g_i^{a_i},$$

with $g_1, \ldots, g_k$ a randomly chosen generator-tuple, as a collision-free hash-function.

Alternatively, it can be used as a means to commit to $k-1$ elements $a_1, \ldots, a_{k-1}$ such that the commitment is unconditionally secure for the committer. This use of the representation problem as a means to commit to multiple numbers has the important feature that one can gradually release certain parts of the information committed to (for example, one can open certain functions of the a_i's) to a verifier with unbounded computing power, without releasing any additional information about the numbers committed to. In a way, one can consider our cash system as being build from commitments on several numbers, partial information about which is gradually opened in the form of certain relations between the numbers. Although strictly speaking it is not correct, due to this viewpoint it benefits the informal discussions to speak of the responses as if they are polynomials. From now on, we will therefore speak of a response that is an index of a representation (e.g. a_1) as being a point, of a response that is linear in the challenge (e.g. $a_i + ca_j \bmod q$) as a line, and in general of a response that is polynomial in the challenge as a polynomial – although the response is actually a value of a constant, linear, or polynomial function in the challenge for one particular challenge.

We discuss ways to do this in detail in Section 8 and further. For now, we remark that the propositions in this section imply that if $(g_1, \ldots, g_k)$ is a randomly chosen generator-tuple, for any number h a polynomially bounded user comes up with, he either knows one representation of h with respect to this tuple, or none at all. If he knows one, he must have taken part in determining h, for example by having chosen an index-tuple $(a_1, \ldots, a_k)$ and computing h as $h = \prod_{i=1}^{k} g_i^{a_i}$. We therefore will speak of this representation as being "the representation" of the user, or "his" representation.

6 Efficiency of computing with the representation problem

Using vector addition chain techniques (see [16] and [28]), one can compute $\prod_{i=1}^{k} g_i^{a_i}$ even for large k almost as efficiently as computing a single exponentiation. We investigate here a new algorithmic idea. The basic steps are as follows:

[Step 1] Receive as input p, q, an index-tuple $(a_1, \ldots, a_k)$ and a generator-tuple $(g_1, \ldots, g_k)$.

[Step 2] Sort the k elements $(g_1, a_1), \ldots, (g_k, a_k)$ in descending order on the a_i's. (The new sequence $(g_1, a_1), \ldots, (g_k, a_k)$ is such that $a_1 \geq a_2 \geq \cdots \geq a_k$.)

[Step 3] For $i = 1$ to $k-1$ do $a_i \leftarrow a_i - a_{i+1},\ g_{i+1} \leftarrow g_i g_{i+1}$. If "stop-condition" holds then output $\prod_{i=1}^{k} g_i^{a_i}$ else go to Step 2.

Note that we use Horner's rule in Step 3 to compute the new g_i's, and we reduce the indices by successively using a generalization of $g^a h^b = (gh)^b g^{a-b}$ (with $a > b$), the basic idea behind Euclid's algorithm for computing gcd's. The sorting algorithm in Step 2 should of course be one suitable for this specific algorithm, since the numbers it has to sort come from an ordered set whose maximum rapidly decreases. In fact, if k is rather large, then many of the a_i will rapidly become 0. In Step 2 we can check for this. Under suitable conditions (e.g. if k is not too large), it may be advantageous to reduce more efficiently using $g^a h^b = (g^{a \operatorname{div} b})^b g^{a \bmod b}$. For very large k however, it will be a waste of time to compute $a \bmod b$. The stop-condition in Step 3 depends on the particular environment the algorithm is implemented in. In general, it will mainly depend on how small the index-tuple has become. The meaning of small itself depends on whether one can apply table lookup or do precomputations and so on.

The resulting algorithm seems quite suitable for use in computing devices with little storage capacity (such as smart cards in off-line electronic cash systems). Basically only the generator-tuple and the index-tuple need to be stored in memory, since in Step 3 the two new tuples can overwrite the old ones.

In the full paper, we fill in all the details we are vague about here, and give an analysis of the running time which for obvious reasons is quite similar to that given in [26] of Euclid's algorithm. We note that the action of first sorting the indices a_i and then forming sequence of successive differences is strongly related to a statistical test for randomness known as the birthday spacings test. We hence can apply some limiting statistics determined for this test (by J. Komlos) to determine how rapidly the sequence of differences collapses.

7 An overview of the literature using the representation problem

The nucleus of Proposition 7 is mentioned without proof in [6] which, as far as we know, is the first article to clarify the status of the representation problem for groups of prime order. In the same paper the authors, as a variation of their basic protocol for proving possession of a discrete logarithm, show a protocol for proving knowledge of a representation (see Section 8). Since then, the representation problem has been used in literature surprisingly little, and no reference is made to [8] anywhere. Corollary 8 can be found in [13], together with a sketch of (a different) proof.

To the best of our knowledge, the representation problem for groups of prime order has furthermore been used as a tool in a handful of articles. For commitment purposes, it was used in [33], [2] and in the confirmation protocol of undeniable signatures ([9, 10]), all with $k = 2$. Furthermore, it has been put to use for signatures unconditionally secure for the signer in [13] and [24] (fail-stop signatures, $k = 2$), and for an identification scheme unconditionally secure for the prover in [32] ($k = 2$). The protocol in this latter article has a striking similarity to the fail-stop signature in [24], the difference being that in the identification scheme one of the two numbers of the public key in the fail-stop signature is chosen randomly by the prover.

Note that when only $k = 2$ is used, as in all the above mentioned articles, Proposition 7 can be proven much easier, and Corollary 8 in fact even becomes superfluous since knowing a representation of h with respect to (g_1, g_2) immediately enables one to compute $\log_{g_1} g_2$. We can hence simply invoke Propositions 2 and 3.

On the basis of Proposition 7, most of the main results in the articles mentioned above can generalized. For example, a straightforward generalization of the protocol of [32] can actually be used to prove knowledge of at least one representation with respect to a generator-tuple of any length. We discuss this in Section 8. As another example, the confirmation protocol of undeniable signatures can be batched (verification of polynomially many signatures in one protocol, rather than just one). Since we do not use this protocol at all for our cash system, but yet think it is a nice result, we include this protocol and the proof in the appendix.

As this paper wishes to show, the applicability of the representation problem is much more general than suggested by the use made of it in the aforementioned articles. This is illustrated already by the fact that for our check extension we use the representation problem for k

$$\begin{array}{lcr}
\text{Prover } \mathcal{P} & & \text{Verifier } \mathcal{V} \\
 & \xrightarrow{(g_1,\ldots,g_k),h} & \\
 & \vdots & \\
w_i \in_{\mathcal{R}} \mathbb{Z}_q & & \\
z \leftarrow \prod_{i=1}^k g_i^{w_i} & \xrightarrow{\quad z \quad} & \\
 & \xleftarrow{\quad c \quad} & c \in_{\mathcal{R}} \mathbb{Z}_q \\
r_i \leftarrow w_i + ca_i \bmod q & \xrightarrow{r_1,\ldots,r_k} & \\
 & & zh^c \stackrel{?}{=} \prod_{i=1}^k g_i^{r_i}
\end{array}$$

FIGURE 1. Proof of knowledge of a representation.

much larger than 2. In particular, its use extends beyond being just a simple tool for e.g., commitments unconditionally secure for the committer.

We discuss the literature on the three basic RSA-variations in Section 17. In this Section, we also sketch how to apply the RSA variations to build an off-line electronic cash system analogous to our discrete log based variant.

8 PROVING KNOWLEDGE OF A REPRESENTATION

In order to allow a polynomial-time prover $\mathcal{P}$ to prove knowledge of a representation (say $(a_1,\ldots,a_k)$) of a number h with respect to $(g_1,\ldots,g_k)$ to a verifier $\mathcal{V}$, we can use a straightforward generalization of the identification protocol described in [32] (see Figure 1). Note that, since the density of representing index-tuples is negligible by Proposition 6, such a protocol certainly makes sense.

Step 1 $\mathcal{P}$ generates at random k numbers $w_1,\ldots,w_k \in_{\mathcal{R}} \mathbb{Z}_q$, and sends $z = \prod_{i=1}^k g_i^{w_i}$ to $\mathcal{V}$.

Step 2 $\mathcal{V}$ generates at random a challenge $c \in_{\mathcal{R}} \mathbb{Z}_q$ and sends it to $\mathcal{P}$.

Step 3 $\mathcal{P}$ computes the responses $r_i = w_i + ca_i \bmod q$, for $i = 1,\ldots,k$, and sends them to $\mathcal{V}$.

$\mathcal{V}$ accepts if and only if $zh^c = \prod_{i=1}^k g_i^{r_i}$, for all $i = 1,\ldots,k$.

PROPOSITION 9 *Under the Discrete Log assumption, the following statements hold:*

1. *(Completeness) If $\mathcal{P}$ is honest (i.e., he knows a representation and follows the protocol), then $\mathcal{V}$ accepts.*

2. *(Soundness) If $\mathcal{P}$ does not know a representation of h with respect to $(g_1, \ldots, g_k)$, there does not exist a strategy for him such that $\mathcal{V}$ accepts with nonnegligible probability of success.*

3. *(Witness hiding) Even with all the information gathered from previous executions of the protocol and unlimited computing power, $\mathcal{V}$ cannot find out any information about the precise representation known by $\mathcal{P}$, if $\mathcal{P}$ followed the protocol in all these executions.*

The proof is similar to that given in [32].

If $\mathcal{V}$ is polynomially bounded in this protocol, he receives information which he couldn't have simulated himself. In order to turn the protocol into one that is zero-knowledge, one can restrict c to a polynomial-sized set and repeat it polynomially many times. In fact, if $c \in_{\mathcal{R}} \{0,1\}$ we precisely get a protocol described in [6].

We now discuss how this protocol can be used to prove certain relations between representations a polynomially bounded prover $\mathcal{P}$ knows of different numbers. To this end, suppose that $\mathcal{P}$ knows one representation $(a_1, \ldots, a_k)$ of h_1, and one representation $(b_1, \ldots, b_k)$ of h_2, both with respect to some generator-tuple $(g_1, \ldots, g_k)$. We suppose $\mathcal{P}$ does not know a representation of 1 with respect to this generator-tuple (which can be accomplished for example by choosing the generator-tuple at random). Rather than showing the general protocol, we show how $\mathcal{P}$ can prove a specific relation, say $na_1 = b_1 \bmod q$, for some $n \in \mathbb{N}$. To this end, we observe that if the relation holds, then

$$h_1^n/h_2 = g_2^{a_2 n - b_2} \cdots g_k^{a_k n - b_k}.$$

Hence, if $\mathcal{P}$ is honest (i.e. the relation $na_1 = b_1 \bmod q$ indeed holds for his representations), he can prove knowledge of a representation of h_1^n/h_2 with respect to $(g_2, \ldots, g_k)$, using the protocol discussed above. However, in case this relation does not hold for his representations, there is no way that he can prove such knowledge. To see this, suppose that $\mathcal{P}$ can have $\mathcal{V}$ accept. Then, by Proposition 9, he must know a representation $(c_2, \ldots, c_k)$ of h_1^n/h_2 with respect to $(g_2, \ldots, g_k)$, i.e.

$$h_1^n/h_2 = g_2^{c_2} \cdots g_k^{c_k}.$$

However, we also have $h_1^n/h_2 = g_1^{a_1 n - b_1} g_2^{a_2 n - b_2} \cdots g_k^{a_k n - b_k}$ from the two representations of $\mathcal{P}$. This implies that

$$g_1^{a_1 n - b_1} g_2^{c_2 - (a_2 n - b_2)} \cdots g_k^{c_k - (a_k n - b_k)} = 1.$$

If $na_1 \neq b_1 \bmod q$, $\mathcal{P}$ therefore knows a non-trivial representation of 1, with respect to $(g_1, \ldots, g_k)$, which contradicts the assumption.

In general, this protocol can be applied to prove that $na_i = b_i \bmod q$ for various i at the same time. If $\mathcal{P}$ wants to prove that e.g. $n_1 a_1 = b_1 \bmod q$, $n_2 a_2 = b_2 \bmod q$, with $n_1 \neq n_2$, the protocol is executed twice (once for $h_1^{n_1}/h_2$, once for $h_1^{n_2}/h_2$).

9 Basic tools for the withdrawal protocol

In our cash system, the identity of a user is encoded entirely in the representation the user knows with respect to a fixed generator-tuple. Due to this approach, we can use a technique that allows us to get rid of the cut and choose methodology that earlier systems use, thus greatly increasing efficiency. This technique is the following. Suppose a user is known to the bank by a pseudonym $m = g_1^{u_1} \cdots g_k^{u_k}$, which links him to his account and his real-life identity. The signature scheme of the bank, whose public key is $(g, h(= g^x))$, is such that the user can do his own blinding of his pseudonym, yet is restricted in the degree of flexibility in doing so. The user will end up with a new number, which is unknown to the bank, together with a valid signature of the bank. Due to the restricted types of manipulations the user can perform in the blinding, a certain structure in the representation the user knows of his pseudonym will remain in the representation the user knows of the new number. This structure is sufficient to distill the users' identity when he double-spends.

Before we discuss the actual blind signature scheme, we clarify this with an example. The user will start at the withdrawal knowing exactly one representation $(a_1, \ldots, a_k)$ of a number m with respect to a generator-tuple $(g_1, \ldots, g_k)$. In our cash system, we accomplish this by having the user choose his own a_i's, and compute m. By Corollary 8, he cannot end up knowing more than one representation. Suppose the blinding were such that the user could only end up with $(m^r, \mathrm{sign}(m))$ if he is to end up with a number which he knows how to represent with respect to the same generator-tuple. The representation he then knows of the new number m^r clearly is simply $(a_1 r, \ldots, a_k r)$, if of course the signature scheme is such that the user cannot compute extra representations of m^r from it. As one sees, the structure which has remained is that induced by taking quotients: for all i, j, the quotient of $a_i r$ and $a_j r$ is the same as that of a_i, a_j (with induced, we mean that this holds for functions like e.g. $(a_1^2 + a_4 a_7)/(a_3 a_5) \bmod q$ in general).

Inspired by this idea, we can encode the user's identity for example as the quotient of say a_i, a_j. During payment, the user will then have to release a line $a_i + a_j c$ in $\mathbb{Z}_q$, with c a challenge of the shop. That is, he opens some partial information about his representation, and then convinces the shop (with a signature) that he did so honestly. When the user double-spends, with overwhelming probability (or without exception if the challenge is forced to be unique for each execution of the payment protocol, which will be the case in a cash system) a_i, a_j can be computed and hence the user's identity. Note however that if the user would be able to blind m to say $m^r g_i^s$, this would no longer be the case. Therefore, we must be able to precisely say to what extend the blinding can be performed by the user. Proposition 12 tells us that the only kinds of blinding that can be performed (if one is to end up with a number of which still a representation is to be known) are those in which m is transformed to $m^r g^s$, with g being the first element of the public key (g, h) of the bank. Therefore, if we make sure that $g_i, g_j \neq g$, we seem to have a workable idea.

Obviously, amongst others the amount of money the information is worth also has to be encoded in some way in the withdrawn information, so the above is a (slight) simplification.

9.1 The restrictive blind signature scheme

We now describe the blind signature scheme we use in the withdrawal protocol. Precisely stated, the purpose of a blind signature protocol is for a receiver to obtain a signature $\mathrm{sign}(m)$ on a message $m \neq 1$ in such a way that not only $(m, \mathrm{sign}(m))$ remains unknown to the signer, but even with infinite computing power it is impossible to link any pair $(m, \mathrm{sign}(m))$ to any specific execution of the protocol. The blind signature scheme we use is actually a signature scheme of [15] (an adaptation from the three-move identification scheme of [36]), although we need some extra features of it. Namely, [15] is not concerned with knowledge of representations of the transformed number, merely with the fact that the blinding (for unconditional unlinkability) can indeed be achieved. For our purposes, this is not sufficient, as we saw in the example: it is essential that there is a restriction on the types of blinding manipulations one can perform (if one is to end up with a number of which a representation is known).

In order to get the reader to understand the blind signature scheme, we first consider some simplifications. We follow the lines of [15] in doing so. Let $g \neq 1, h(= g^x) \neq 1$ be the public key of the signer, and $m \neq 1$ a message from the receiver. The signer is supposed to sign m with $z(= m^x)$ and a signed proof that $\log_g h = \log_m z$ (note that z by itself is an undeniable signature on m). This can be accomplished with the following protocol.

In Step 1, the signer generates at random a number $w \in_{\mathcal{R}} \mathbb{Z}_q$, and sends $z = m^x$, $a = g^w$ and $b = m^w$ to the receiver. In Step 2, the receiver generates at random a challenge $c \in_{\mathcal{R}} \mathbb{Z}_q$ and sends it to the signer. In Step 3, the signer sends the response $r = w + cx \bmod q$ to the receiver. The receiver accepts if and only if $h^c a = g^r$ and $z^c b = m^r$. Note that the only difference so far with the scheme of [36] is that, in Step 1, two extra numbers z and b are sent along. Using the technique introduced in [21], the receiver can transform this minimum-knowledge protocol into a signature scheme by computing c as $\mathcal{H}(m, z, a, b)$, with $\mathcal{H}(\cdot)$ a suitable (see [15]) one-way hash-function. Since this c can just as well be computed by the signer, actually only one transmission from the signer to the receiver is needed then, consisting of the signature $\text{sign}(m) = (z, a, b, r)$ on m. The signature is valid if and only if the relations $h^c a = g^r$ and $z^c b = m^r$ hold, with c computed by a verifier from the signature as $\mathcal{H}(m, z, a, b)$. There exist exponentially many such signatures on m, depending on the choice of w made by the signer, but their density among all possible four-tuples of the above form is negligible. In fact, the assumption concerning $\mathcal{H}(\cdot)$ implies that one cannot generate pairs $(m, \text{sign}(m))$, i.e., existential forgery of pairs is infeasible (see [15]).

Clearly, this scheme is certainly not a blind signature scheme because one can easily match a pair $(m, \text{sign}(m))$ with a specific execution of the signature protocol (m as well as all numbers in $\text{sign}(m)$ remain completely unchanged in the protocol). To achieve unconditional unlinkability, the receiver in the above protocol determines a new message m', new numbers a', b' and z', and sends a blinded version c of $c' = \mathcal{H}(m', z', a', b')$ to the signer. The signer responds with $r = w + bc \bmod q$, which the receiver transforms in some way to a number r' such that (z', a', b', r') is a valid signature on m'. Observe that in fact also w must be transformed to some w' (whereas it is unknown to the receiver), because the signer is given unconditional computing power and could otherwise still link by computing w from the signature and comparing it to all his executions of the protocol. With this in mind, we now show the actual blind signature protocol (see Figure 2).

Step 1 The signer generates at random a number $w \in_{\mathcal{R}} \mathbb{Z}_q$, and sends $z = m^x$, $a = g^w$ and $b = m^w$ to the receiver.

Step 2 The receiver generates at random four numbers s, t and u, v (all in $\mathbb{Z}_q$). Using the first two of these numbers, he computes $m' = m^s g^t$. With the other two, he computes new numbers a', b' and z' as follows (with $w' = uw + v \bmod q$): $a' = a^u g^v \, (= g^{w'})$, $b' = a^{ut} b^{us} (m')^v \, (= (m')^{w'})$ and $z' = z^s h^t \, (= (m')^x)$. He then computes $c' = \mathcal{H}(m', z', a', b')$ and sends $c = c'/u \bmod q$ to the signer.

Step 3 The signer responds with $r = w + cx \bmod q$.

Receiver		Signer
		$(g, h = g^x)$
	$\xrightarrow{\quad m \quad}$	
		$w \in_{\mathcal{R}} \mathbb{Z}_q$
		$z \leftarrow m^x$
		$a \leftarrow g^w$
	$\xleftarrow{\quad z, a, b \quad}$	$b \leftarrow m^w$
$s, t \in_{\mathcal{R}} \mathbb{Z}_q$		
$m' \leftarrow m^s g^t$		
$u, v \in_{\mathcal{R}} \mathbb{Z}_q$		
$a' \leftarrow a^u g^v$		
$b' \leftarrow a^{ut} b^{us} (m')^v$		
$z' \leftarrow z^s h^t$		
$c' \leftarrow \mathcal{H}(m', z', a', b')$		
$c \leftarrow c'/u \bmod q$	$\xrightarrow{\quad c \quad}$	
	$\xleftarrow{\quad r \quad}$	$r \leftarrow w + cx \bmod q$
$h^c a \stackrel{?}{=} g^r$		
$z^c b \stackrel{?}{=} m^r$		
$r' \leftarrow ur + v \bmod q$		

FIGURE 2. Blind signature protocol

As before, the receiver accepts if and only if $h^c a = g^r$ and $z^c b = m^r$. Note that (z, a, b, r) itself is not a correct signature on m, since r is a response to a challenge different from $\mathcal{H}(m, z, a, b)$. However, the receiver can compute $r' = ur + v \bmod q$ and, as easily can be verified, ends up with a valid signature $\text{sign}(m') = (z', a', b', r')$ on m'. This is a digital signature, which can be verified by anyone.

It is clear that if the receiver follows the protocol, he ends up with a number m' plus signature, such that he knows a representation of m' with respect to (m, g) (namely (s, t)).

PROPOSITION 10 *The following statements hold:*

1. *If the receiver follows the protocol, then pairs $(m, sign(m))$ obtained via this signature protocol are unconditionally unlinkable to any specific execution of the protocol.*
2. *Even if the signer would know the m' obtained by the receiver, the precise representation (s, t) with respect to (m, g) known by the receiver is unconditionally hidden among the set of all representations of m'.*

SKETCH OF PROOF. Given any $h \neq 1$, for each pair $(m, \text{sign}(m))$ and the information that a signer gets during the execution of any protocol in which the receiver accepts, there are exactly q possible random choices of sets (s, t, u, v) that could have been made by the receiver that bring about the link. The reader is referred to [15] for the precise proof. The second statement is trivial. $\square$

The following proposition tells us what kind of blinding manipulations can be feasibly performed, under the condition that one must end up with a number (signed by the bank) of which one knows a representation with respect to (m, g). We first give a lemma.

LEMMA 11 *Under the Diffie-Hellman assumption there does not exist a polynomial-time algorithm A that, on input a randomly chosen triple $g_1, g_2, (g_1 g_2)^w$, outputs g_1^w with nonnegligible probability of success.*

PROOF. Using algorithm A as a subroutine we construct the following algorithm, which takes as inputs $g_1 (= g^a), g_2 (= g^b)$ for random $a, b \in \mathbb{Z}_q$:

[Step 1] Generate at random an element $g \in G_q \setminus \{1\}$, and feed $g_1, g/g_1, g_2$ into algorithm A.

[Step 2] Output the output of algorithm A.

If one writes $h = (g_1 g_2)^w$, then it is clear that A, on inputs g_1, g_2, outputs $g_1^{\log_{g_1 g_2} h}$. Observe that since g is randomly chosen in Step 1, g_1 and g/g_1 are independently distributed. Therefore, the algorithm we just constructed outputs $(g^a)^{\log_g g^b} = g^{ab}$ with nonnegligible probability of success, i.e. we have constructed a polynomial-time algorithm that with nonnegligible probability of success outputs Diffie-Hellman keys. This contradicts the Diffie-Hellman assumption. $\square$

Clearly, if one can break the Diffie-Hellman assumption then one can compute g_1^w from $g_1, g_2, (g_1 g_2)^w$, and hence the two problems are actually polynomial-time equivalent.

The following proposition is essential for proving the security of our cash system.

PROPOSITION 12 *Let m, g be elements of G_q (as in the blind signature protocol), such that the receiver does not know $\log_g m$. Then the following two statements hold:*

1. *Under the Discrete Log assumption, there is no strategy that has nonnegligible probability of success for the receiver such that he ends up with a pair $(m', sign(m'))$ for which he knows two different representations of m' with respect to (m, g).*

2. *Let $g' \neq 1$ be randomly chosen, independently from m and g, and assume that the hash-function $\mathcal{H}(\cdot)$, in addition to being as in [15], is collision-free. Under the Diffie-Hellman assumption, there is no strategy for him that has nonnegligible probability of success of ending up with a pair $(m', sign(m'))$ for which he knows a representation $(s, t, t' \neq 0)$ of m' with respect to (m, g, g').*

Both statements hold even if the receiver has access to polynomially many transcripts of previous executions in which he played the role of the receiver.

SKETCH OF PROOF. (1) The first statement follows from the fact that being able to end up with $(m, \text{sign}(m))$ such that two different representations (s, t) and (s', t') of m are known implies that $\log_g m = (t' - t)/(s - s') \bmod q$. However, the signer himself need not know $\log_g m$ in order to perform the blind signature protocol and, in particular, could just as well be polynomial-time.

(2) The second statement follows from the fact that in order to obtain a valid signature $\text{sign}(m) = (z, a, b, r)$ on a message m, the third component b of the signature must equal m^w, by assumption on the collision-freeness of the hash-function $\mathcal{H}(\cdot)$. Therefore, if the receiver can end up with b' corresponding to m' of the form $m' = m^s g^t (g')^{t'}$ (with $t' \neq 0$), he can compute the undeniable signature $(g')^w$ on g', since $(g')^w = (b'/(a^t b^s)^{1/t'}$ (note that this holds regardless of the distribution with which (s, t, t') is generated by the receiver).

To see how difficult this task is, we have to know what information the receiver has at his disposal to achieve this. First, observe that c must be determined by the receiver before he receives the response $w + cx \bmod q$, hence only the information in the first two transmissions of this execution can be of help, and that of previous executions. However, since w is chosen uniformly at random by the signer in each execution of the protocol, previous executions of the protocol cannot help the receiver in determining $(g')^w$ since w is uncorrelated to x and the choices of w in previous executions (and hence to the responses in those executions), and so the receiver might as well have simulated all the choices of the signer in those previous executions. For the same reason, from the specific execution in which the receiver is to come up with $(g')^w$, the numbers $h (= g^x)$ and $z (= m^x)$ cannot help him.

Furthermore, since g' is randomly chosen, independent from m, g, the receiver does not know how to represent g' with respect to (m, g). Therefore, it remains to prove that under the Diffie-Hellman assumption it is infeasible to compute $(g')^w$ given g, g^w, m, m^w, g', with g' such that no representation (a, b) of g' is known with respect to (m, g). But this is easy to prove. To see this, note that the existence of a polynomial-time algorithm to compute $(g')^w$ on inputs the same inputs (i.e. g, g^w, m, m^w, g') and in addition also $\log_g m$, implies the existence of a polynomial-time algorithm that, on inputs g, g^w, g', outputs $(g')^w$ (since the inputs m, m^w can then be simulated because g' and g are independently distributed from m). This latter algorithm in turn is trivially convertible to an algorithm that solves the Diffie-Hellman assumption, in fact it already is one (see Section 18). That is, there does not exist a polynomial-time algorithm that, on inputs $g, g^w, m, m^w, g', \log_g m$, outputs $(g')^w$, if and only if the Diffie-Hellman assumption holds. As a consequence, no polynomial-time algorithm can exist that outputs $(g')^w$ on inputs g, g^w, m, m^w, g' (since it receives even less information).

Therefore, assuming the Diffie-Hellman assumption, if and only if the receiver knows a representation of g' with respect to (m, g), he can feasibly compute $(g')^w$. This completes the proof. □

For the security of our withdrawal scheme, this proposition it not sufficient. However, in that case we can use two extra requirements concerning the representation a user begins with, and the representation he is to end with (see Section 12).

From now on, we implicitly assume that the hash-function $\mathcal{H}(\cdot)$ in addition to satisfying the requirements of [15] is collision-free. This requirement is only to ensure that we can apply the second statement of Proposition 12 when we want to prove that the receiver in the protocol cannot obtain certain representations. The requirement of [15] is to ensure that existential forgery of pairs m, sign(m) is infeasible. We do not know whether the assumption that $\mathcal{H}(\cdot)$ be collision-free can be weakened significantly. However, the existence of a collision-free hash-function is a weaker assumption then the Diffie-Hellman assumption whereas breaking the security of our system is no harder than breaking the Diffie-Hellman assumption, so weakening the assumption on the hash-function does not increase the security of our system. Note that in Section 5 we described a hash-function that is collision-free if and only if the Discrete Log assumption is true.

We apply Proposition 12 to our cash system as follows. At the start of the withdrawal protocol, suppose the user knows exactly one representation of m with respect to a generator-tuple $(g_1, \ldots, g_k)$, say $(a_1, \ldots, a_k)$ (if he does not know any, the entire argument becomes even more simple). A sufficient condition to ensure that the user does not know more than one representation is that he does not know a representation of 1 with respect to $(g_1, \ldots, g_k, g)$. This is easily taken care of by the bank by for example generating the $k+1$-tuple $(g_1, \ldots, g_k, g)$ at random.

During the protocol, the user blinds m to m', which is the number that he will receive a signature on. As we saw in the proof of the second statement of Proposition 12, the only way in which he can do this is by choosing numbers s, t, t' and g', and computing $m' = m^s g^t (g')^{t'}$. If $t' \neq 0$, in addition he must know a representation (x, y) of g' with respect to (m, g). It is easy to show that the numbers g' for which this latter requirement is met are precisely those g' of which the user knows a representation $(b_1, \ldots, b_{k+1})$ with respect to $(g_1, \ldots, g_k, g)$ for which $b_i = x a_i \bmod q$ (for $1 \leq i \leq k$) and $b_{k+1} = y$. From this it is clear that the user might as well have computed m' as $m' = m^{s+x} g^{t+y}$.

We give a small example of this. Suppose the user knows (a_1, a_2) such that $m = g_1^{a_1} g_2^{a_2}$, but does not know a representation of g with respect to (g_1, g_2). He can then compute $m' = m^s g^t = g_1^{a_1 s} g_2^{a_2 s} g^t$ for random choices of s, t. Suppose he can compute m' (and obtain a signature) as e.g. $m' = m^s g^t (g_1 g)^{t'}$: he would then have to know by Proposition 12 a pair (x, y) such that $g_1 g = m^x g^y$. But then $(a_1 - 1, a_2, y - 1)$ is a representation of 1. Since this must be trivial (not even the signer need know anything more), this imposes the condition $a_1 = 1, a_2 = 0$, and $y = 1$. That is, $m = g_1$, and the user might as well have computed m' as $m' = m^{s+1} g^t$ instead.

Observe that, in general, the user therefore always ends up with a signed number m' for which the representation $(b_1, \ldots, b_k, c)$ he knows of m' with respect to $(g_1, \ldots, g_k, g)$ is always such that $b_i = a_i s$ (with $1 \leq i \leq k$) and $c = t$ for some pair (s, t). As one sees, $b_i a_j = a_i b_j$ for all i, j. The fact that certain relations between the indices of the representation the user knows of m' are precisely the same as of the representation of m is a crucial point in our system.

In the protocols for the cash system (including the extension to checks) which we discuss in Section 11, the blinding manipulations that the user could actually perform if he is to be able to pay at a shop with this information, is even more restricted in that he can only transform m to $m' = m^s$ (i.e. $t = 0$). This is due to the fact that the payment protocol requires him to know a representation of m' with respect to $(g_1, \ldots, g_k)$, rather than $(g_1, \ldots, g_k, g)$. If he would transform it to $m' = m^s g^t$ such that he knows $(s, t \neq 0)$, it is easy to show that he ends up with m' which he does not know how to represent with respect to m.

Because of the above discussion, the second statement of Proposition 12 is crucial to the security of our cash system, and is why the system can be broken if one can break the Diffie-Hellman assumption (the identity of the double-spender, which is encoded in the structure that remains in the representations, then no longer is revealed since the structure is destroyed). We here note that if there would exist a polynomial-time algorithm that with nonnegligible probability of success computes Diffie-Hellman keys, by the Decision Diffie-Hellman assumption (to be discussed in Section 18) one cannot recognize a correct answer. Yet it means that having access to such an algorithm, one has a nonnegligible probability of success of receiving a pair m, sign(m), which allows one to double-spent this information without being identified.

Observe that if in the first transmission of the blind signature protocol $m^{-s}g^{-t}$ is sent, then the signature scheme allows the receiver to end up with a signature on a number m of his own choice. In our cash system, m will never be a choice of the user, but it is more or less fixed in advance. A similar thing holds true for the extension to checks. Note that in the withdrawal protocol of previous systems the information to be signed is sent in an blinded form, and then unblinded after the signature has been made, whereas in our withdrawal protocol almost the opposite happens. This has the extra advantage that it is infeasible for users to obtain blind signatures on arbitrarily chosen messages (since, given some fixed numbers m, g, and a "target" message n, one cannot compute (s, t) such that $n = m^s g^t$). Compare this to the RSA blind signature scheme of [11], in which the user sends $r^v \mathcal{H}(m)$ mod n, the bank supplies the v-th root of this number, and the user transforms it to the pair $(m, \mathcal{H}(m)^{1/v})$: the user can get a signature on any message he likes, which clearly can only help in attacking the system by multi-user and/or chosen message attacks. Not being able to choose freely the message that will be signed also has the advantage that multi-user attacks have less probability of being successful. So the restrictive blind signature scheme that we use also helps us in proving correctness of the system.

10 Basic tools for the payment protocol

In order to prevent double-spending of a coin (or check), the user at the shop has to reveal some information that has the property that one such piece of information does not reveal any Shannon information about his identity, whereas knowledge of two such pieces enables one to extract this identity in polynomial time (one-show signature). As we explained in Section 3, the technique we apply for this is based on the simple fact that the slope of a line is uniquely determined by two points, yet completely indeterminate from one. In the previous section we gave an example of how we use this technique in conjunction with the withdrawal protocol. Since a polynomial of degree k is uniquely identified by $k+1$ different points, we can generalize this to allowing the users to spend their money k times. In general case, k spendings of the same user are untraceable to him, but the payments themselves are linkable. We note that in case a user pays say \$10 with a 10-show coin worth \$1 (i.e. in the same transaction), this is no problem at all, whereas it is more efficient than using a check of which the unspent part then must be refunded. It is also conceivable that users that are particularly keen on their privacy only pay with k-show information in transactions of which they are sure not to be identified by some external means.

There are various ways in which the payment protocol can be achieved: in essence, we have to come up with a challenge-response protocol of which the response is a line (a polynomial in general). Furthermore, a valid transcript of the payment protocol must be unforgeable in polynomial time, since shops should not be able to deposit transcripts they forged. That is, the payment protocol has to consist of a signature, and it must be such that the user can release points (for the check extension), lines, and in general polynomials, and prove with a signature that no bogus information was given.

We first show a simple protocol to randomize a representation. We then discuss two basic methods that enable the prover to reveal partial information about the representation he knows in the form of a line. The second of these has the nice property that the responses of the prover at the same time constitute a signature on the challenge (we call this property self-certifiability of the responses), and hence uses only three moves. The first method needs two more moves, but has the advantage that it can also be used for minimum-knowledge proofs (the responses are not self-certifiable), and is more efficient for responses that are polynomials of high degree.

We then consider some how to use method 1 or 2. Finally, it is shown how these protocols can be used to simultaneously reveal partial information of knowledge in the form a bunch of points, lines, and polynomials of higher degree.

10.1 Randomization of representations

In the set-up of our cash system, the user ($\mathcal{P}$, polynomially bounded) must establish an account with the bank. This will consist of a number $h = g_1^{a_1} g_2^{a_2}$, such that (a_1, a_2) is unconditionally hidden in the set of all representations of h with respect to (g_1, g_2). In some situations, the bank ($\mathcal{V}$, unbounded) may want each element of the representation known to $\mathcal{P}$ to be mutually random. In general, $\mathcal{V}$ may want to establish in cooperation with $\mathcal{P}$ a number h and a representation of h with respect to a generator-tuple $(g_1, \ldots, g_k)$ such that each element of the representation (and hence h) is mutually random. To this end, let g_{k+1} be a generator distinct from the others. Then the following simple protocol can be used:

Step 1 $\mathcal{P}$ generates uniformly at random an index-tuple $(x_1, \ldots, x_{k+1})$, and sends $h' = \prod_{i=1}^{k+1} g_i^{x_i}$ to $\mathcal{V}$.

Step 2 $\mathcal{V}$ sends a randomly chosen index-tuple $(y_1, \ldots, y_k)$ to $\mathcal{P}$.

Step 3 $\mathcal{P}$ sends x_{k+1} to $\mathcal{V}$.

If $\mathcal{P}$ was honest in Step 3, $(x_1 + y_1 \bmod q, \ldots, x_k + y_k \bmod q)$ is a representation of $h = (\prod_{i=1}^{k} g_i^{y_i}) h' / g_{k+1}^{x_{k+1}}$ with respect to $(g_1, \ldots, g_k)$, which satisfies the above requirement. It is easy to show that the only way in which $\mathcal{P}$ can end up knowing a representation of h is if he follows the protocol.

10.2 Method 1.

Before giving the general protocol, we start with an example which comes close to the actual payment protocol we use in our basic cash system.

Suppose $\mathcal{P}$ has obtained a signature of the bank on a number $g_1^{a_1} g_2^{a_2} g_3^{a_3}$. The partial information $\mathcal{P}$ will have to reveal is the response $r = a_1 + a_2 c \bmod q$, with c being a challenge from a verifier $\mathcal{V}$ (who is not necessarily polynomially bounded). In our cash system the identity of $\mathcal{P}$ is a certain relation between a_1, a_2 and a_3, namely $(a_1/a_3 \bmod q, a_2/a_3 \bmod q)$. For simplicitly, we assume the identity here to be $a_1/a_2 \bmod q$. Since $\mathcal{V}$ has no way to convince himself of the fact that $\mathcal{P}$ gave the correct response, some protocol is needed which enables $\mathcal{P}$ to prove this to $\mathcal{V}$, without revealing any Shannon information about $a_1/a_2 \bmod q$ (see Figure 3).

Step 1 $\mathcal{V}$ generates at random a challenge $c \in_{\mathcal{R}} \mathbb{Z}_q$ and sends this to $\mathcal{P}$.

Step 2 $\mathcal{P}$ computes the response $r = a_1 + a_2 c \bmod q$ and sends this to $\mathcal{V}$.

Step 3 $\mathcal{P}$ proves knowledge of a representation of h/g_1^r with respect to $(g_2/g_1^c, g_3)$.

$\mathcal{V}$ accepts if and only if he accepts $\mathcal{P}$'s proof of knowledge in Step 3.

In Step 3, any protocol for proving knowledge of a representation can be used (we can use the one described in Section 8), as long as it does not reveal any information about the specific representation known by $\mathcal{P}$. However, when used in a payment protocol, it must actually be a signature. Since the shop will send the transcript of this protocol to the bank, who has to be able to verify its validity and be assured that is is not a forgery, in a payment protocol h is accompanied by a signature of the bank. This will guarantee that transcripts of the payment protocol cannot be forged existentially. We discuss this further in Section 12.

PROPOSITION 13 *Under the Discrete Log assumption, the following statements hold:*

1. *(Completeness) If $\mathcal{P}$ knows a representation of h and follows the protocol, $\mathcal{V}$ accepts.*
2. *(Soundness) Suppose $\mathcal{P}$ knows at most one representation of h at the start of the protocol. Then $\mathcal{P}$ cannot determine an incorrect response r in Step 2, such that there exists a strategy for him leading $\mathcal{V}$ to accept with nonnegligible probability of success.*
3. *(Witness hiding) If $\mathcal{P}$ follows the protocol, then $\mathcal{V}$ cannot obtain any Shannon information about $a_1/a_2 \bmod q$ (the identity) from one execution of the protocol.*

SKETCH OF PROOF. First, note that if $r = a_1 + a_2 c \bmod q$, then $h/g_1^r = (g_2/g_1^c)^{a_2} g_3^{a_3}$. Hence, if $\mathcal{P}$ is honest, (a_2, a_3) is a representation of h/g_1^r with respect to $(g_2/g_1^c, g_3)$. This proves the completeness property of the protocol.

Suppose $\mathcal{P}$ is able to come up with a response r such that he can successfully convince $\mathcal{V}$ in Step 3 (i.e., $\mathcal{V}$ accepts). Then, according to Proposition 9, he must know a representation in

$$
\begin{array}{lcl}
\text{Prover } \mathcal{P} & & \text{Verifier } \mathcal{V} \\
(h = \prod_{i=1}^{3} g_i^{a_i}) & & \\
& \xrightarrow{\quad h, (g_1, g_2, g_3) \quad} & \\
& \vdots & \\
& \xleftarrow{\qquad c \qquad} & c \in_{\mathcal{R}} \mathbb{Z}_q \\
r \leftarrow a_1 + a_2 c \bmod q & \xrightarrow{\qquad r \qquad} & \\
& \xleftrightarrow{\text{(proof of correctness)}} &
\end{array}
$$

FIGURE 3. Partial release of knowledge

Step 3 of h/g_1^r with respect to $(g_2/g_1^c, g_3)$, say (u_1, u_2). More specifically, $\mathcal{P}$ knows r, u_1 and u_2 such that $h/g_1^r = (g_2/g_1^c)^{u_1} g_3^{u_2}$, implying that he knows r, u_1, u_2 such that $h = g_1^{r-u_1c} g_2^{u_1} g_3^{u_2}$. Under the assumption that $\mathcal{P}$ knows exactly one representation of h, if $r \neq a_1 + a_2 c \bmod q$ then apparently $\mathcal{P}$ knows two different representing index-tuples of h with respect to (g_1, g_2, g_3). According to Corollary 8, this contradicts the Discrete Log assumption. If $\mathcal{P}$ did not know any representation at all, then apparently he knows one now, contradicting Proposition 7.

To prove the third statement, observe that in Step 3 no information about the representation of h/g_1^r with respect to $(g_2/g_1^c, g_3)$ is revealed. Since $\mathcal{P}$ is honest, this representation is (a_2, a_3). Furthermore, r and h together also do not reveal any information on either of a_1, a_2 and a_3, because for each guess of $\mathcal{V}$ for a_1', a_2' (such that $r = a_1' + a_2' c \bmod q$), there is exactly one a_3' such that $h = \prod_{i=1}^{3} g_i^{a_i'}$. □

However, if $\mathcal{P}$ performs this protocol twice with $\mathcal{V}$, then with overwhelming probability a_1, a_2 (and $g_3^{a_3}$, but we do not need this) is revealed, and the identity $a_1/a_2 \bmod q$ of the double-spender can be computed.

We can generalize this protocol straightforwardly in various ways. For example, the protocol can be adapted in such a way that the identity a_1 of $\mathcal{P}$ knowing a representation $(a_1, \ldots, a_k)$ of h with respect to $(g_1, \ldots, g_k)$, is revealed after $j+1$ executions of the protocol $(2 \leq j < k)$ but remains unconditionally hidden after up to j executions (j-show signatures). To this end, the response must simply be a polynomial of degree j.

In Proposition 13, the witness hiding property obviously extends beyond just $a_1/a_2 \bmod q$. However, it is not true for just any function of the a_i's. For example, it is clearly not true for $a_1 + a_2 \log_{g_1} g_2 + a_3 \log_{g_1} g_3$, since this can be extracted by the receiver even without any execution of the protocol. For the cash system, this means that, in order to apply this proposition, the identity has to be encoded as a function of the a_i's in a suitable way, such that one execution of this protocol still does not release any Shannon information about the identity. In the full paper, we will formalize this using a set of so-called valid identity-encodings. This is the set of all functions of the a_i's satisfying this witness-hiding property. Note that the definition of this set clearly is dependent on the particular protocol under consideration.

10.3 Method 2.

We now describe another method to achieve the same goal. Because we assume the previous discussion explains the basic idea, we right away show the most general version of the protocol, relating to $(k-1)$-show signatures. To this end, suppose that there are k numbers $h_1, \ldots, h_k$, and $\mathcal{P}$ knows how to write them as $h_1 = g_1^{a_{11}} \ldots g_l^{a_{1l}}$, $h_k = g_1^{a_{k1}} \ldots g_l^{a_{kl}}$. The identity of $\mathcal{P}$ again is encoded using a function from the set of suitable identity-encodings. The elements $h_1, \ldots, h_k, g_1, \ldots, g_k$ are also known to $\mathcal{V}$ before the protocol starts (see Figure 4).

Step 1 $\mathcal{V}$ generates at random a challenge (message) $c \in_{\mathcal{R}} \mathbb{Z}_q$ and sends this to $\mathcal{P}$.

Step 2 $\mathcal{P}$ computes the l responses $r_1, \ldots, r_l$ as

$$r_i = a_{i1} + a_{i2}c + \cdots + a_{ik}c^{k-1} \bmod q$$

and sends these to $\mathcal{V}$.

$\mathcal{V}$ accepts if and only if

$$\prod_{i=1}^{k} h_i^{c^{i-1}} = \prod_{i=1}^{l} g_i^{r_i}.$$

This protocol can be viewed as a generalization of a fail-stop signature protocol of [24] (or of [32], for that matter), although we use it in yet another way. This is already obvious from the property that in the cash system users remain anonymous, and hence we in fact even cannot use the special properties of fail-stop signatures (we prevent framing in another way). Instead, we use this protocol for our cash system as a means for the prover to release lines (we extend this in the next Subsection). As the next proposition will show, there are no extra moves necessary to prove correctness of the revealed information, since the responses are self-certifiable. In contrast, the response in method 1 is not self-certifiable (there is no verification relation), hence the certification must be provided interactively, for which extra moves are needed. Note however that in order to reply with a polynomial of degree k, in method 2, k numbers are needed in the first move (for self-certifiability), whereas with method 1 this is independent of k.

Although the protocol of method 2 has the self-certifiability property, it cannot be used in our cash system precisely as described here, since transcripts can easily be forged (by the shop) by choosing numbers for the second and third moves, and computing the corresponding numbers of the first move according to the verification relation. In our cash system, the numbers in the first transmission are accompanied by a signature of the bank, which guarantees that transcripts cannot feasibly be forged. Since shops also have to be able to verify whether a user is giving them validated cash, this signature has to be included anyway. This remark also applies to method 1.

Prover $\mathcal{P}$		Verifier $\mathcal{V}$
	$\xrightarrow{h_1, \ldots, h_k, g_1, \ldots, g_k}$	
	$\vdots$	
	$\xleftarrow{c}$	$c \in_{\mathcal{R}} \mathbb{Z}_q$
$r_i \leftarrow \sum_{j=1}^{k} a_{ij} c^{j-1} \bmod q$	$\xrightarrow{r_1, \ldots, r_l}$	
		$\prod_{i=1}^{k} h_i^{c^{j-1}} \stackrel{?}{=} \prod_{i=1}^{l} g_i^{r_i}$

FIGURE 4. Partial release of knowledge

For the next proposition, we assume that the identity is encoded using a function from the set of valid identity-encodings. Again, we formally define this set in the full paper.

PROPOSITION 14 *Under the Discrete Log assumption, the following two statements hold:*

1. *$\mathcal{P}$ cannot determine any of the responses r_i in Step 2 to be incorrect, in such a way that $\mathcal{V}$ accepts with nonnegligible probability.*
2. *If $\mathcal{P}$ is honest, there is no way for any $\mathcal{V}$ to obtain any Shannon information about the identity of $\mathcal{P}$ from $k-1$ executions of the protocol.*

The proof is quite simple and can be found in the full paper.

Responses to k different challenges reveal all the a_{ij}. That is, if $\mathcal{P}$ executes this protocol k times with $\mathcal{V}$, his identity is revealed. Note that, in contrast to method 1, not only his identity is revealed, but his entire knowledge (all the a_i's) is revealed.

10.4 Method 1 versus method 2

In our basic cash system (discussed in the next section), we use the protocol of method 2, with $k = 2$, $l = 3$, and use an extra technique to increase efficiency. After performing the withdrawal protocol, the user ends up with a number $m = g_1^{a_1} g_2^{a_2} g_3^{a_3}$ (plus the signature of the bank), together with the representation (a_1, a_2, a_3). His identity is encoded as some some suitable (i.e. from the set of valid identity-encodings) function of a_1, a_2, a_3. He then determines a splitting of m into two (k in general for a $(k-1)$-show signature) different parts, say A and B. If he is to know a representation of both A and B with respect to (g_1, g_2, g_3) (say (x_1, y_1, z_1) and (x_2, y_2, z_3), respectively), then the following relations must necessarily hold for these representations:

$$x_1 + x_2 = a_1 \bmod q, \; y_1 + y_2 = a_2 \bmod q, \; z_1 + z_2 = a_3 \bmod q,$$

otherwise we would have a contradiction with Proposition 7 (or Corollary 8, for that matter). Double-spending reveals all the x_i's, y_i's and z_i's, and hence a_1, a_2 and a_3. Therefore the bank can compute the identity of the double-spender.

We of course have to make sure that the user commits himself to the specific splitting (A, B), for otherwise he could make a new splitting for each payment and his identity would never be revealed. We achieve this by having the bank during the withdrawal put his signature on A (and hence B) as well (without getting to know A and B). What then actually happens is that the user gets a signature on (A, B). This holds for a setting in which a coin can be spent $k-1$ times as well: using the splitting technique in conjunction with the protocol of method 2, the user in the withdrawal phase ends up with $m = \prod_{i=1}^{k} A_i$, plus a signature on $(A_1, \ldots, A_k)$.

It is important to note that if it weren't for this splitting trick, in general it would be more efficient to use the protocol of method 1: in method 2, actually the only reason why k numbers $(h_1, \ldots, h_k)$ are used (instead of just one, as in method 1) is to achieve the self-certifiability property for responses that are polynomials of degree k. Without the splitting technique, the withdrawal protocol would in effect have to be performed k times (once for each number m_i), and it would have to be clear from each of the k signatures (in the interest of e.g. the shops) that they belong to the same withdrawn ensemble. Although the withdrawal could be done in parallel, it would obviously be quite inefficient (unless k is very small). In our cash system (including the extension to checks), the splitting technique can be applied without difficulties, and so we use method 2 even for the multi-show extensions.

10.5 Revealing bunches of points, lines, and polynomials

We now show how to extend both methods in such a way that the prover can release a bunch of points, lines, and polynomials. The ability to release both points and lines is needed for the extension of our system to checks and divisibility.

The basic idea is very simple and can be applied to all the tools we described. Assume a polynomially bounded $\mathcal{P}$ has a number m and knows one representation $(a_1, \ldots, a_k)$ of m with respect to a generator-tuple $(g_1, \ldots, g_k)$. Consider the case where $\mathcal{P}$ wants to release some of the a_i's (points) of his representation to $\mathcal{V}$. Before the protocol starts, $\mathcal{P}$ just sends them to $\mathcal{V}$ and the rest of the protocol is then performed with the products of the $g_i^{a_i}$ for which the a_i's have been revealed, divided out first. For example, in the general protocol of method 1, if $\mathcal{P}$ first opens $a_l, \ldots, a_k$ for some $l < k$, the protocol remains the same, with the new h being $h / \prod_{i=l}^{k} g_i^{a_i}$, and the new generator-tuple $(g_1, \ldots, g_{l-1})$.

We illustrate this with an example. Suppose $\mathcal{P}$ wishes to reveal a_1 and a_2 of his representation of m. We note that $m/(g_1^{a_1} g_2^{a_2}) = \prod_{i=3}^{k} g_i^{a_i}$, hence in case $\mathcal{P}$ honestly computed his responses r_1, r_2 as $r_1 = a_1$, $r_2 = a_2$, he knows how to represent $m/(g_1^{r_1} g_2^{r_2})$ with respect to $(g_3, \ldots, g_k)$. Vice versa, if $\mathcal{P}$ was dishonest ($r_1 \neq a_1$ or $r_2 \neq a_2$), he cannot prove knowledge of $m/(g_1^{r_1} g_2^{r_2})$ with respect to $(g_3, \ldots, g_k)$, since he must then know a representation (by definition), say $(c_3, \ldots, c_k)$, of $m/(g_1^{r_1} g_2^{r_2})$ with respect to $(g_3, \ldots, g_k)$. This however implies that $\mathcal{P}$ knows the representations $(r_1, r_2, c_3, \ldots, c_k)$ and $(a_1, \ldots, a_k)$ of m with respect to $(g_1, \ldots, g_k)$. If $r_1 \neq a_1$ or $r_2 \neq a_2$, then these two representations are distinct, which is in contradiction with the assumption. Note that if $\mathcal{P}$ originally knew no representation of m at all, he cannot succeed either in proving knowledge.

In general, $\mathcal{P}$ can release certain points a_i, with $i \in S \subseteq \{1, \ldots, k\}$. If and only if $\mathcal{P}$ honestly reveals these points (i.e. $r_i = a_i$ for $i \in S$), he knows a representation of $m/\prod_{i \in S} g_i^{r_i}$ with respect to the tuple of generators g_i for which $i \notin S$. For the checks and the divisibility we use this in the following way.

The user $\mathcal{P}$ sends to $\mathcal{V}$ (the shop) the numbers A, B, plus the signature of the bank on these numbers. $\mathcal{P}$ knows representations $(a_1, \ldots, a_k)$, respectively $(b_1, \ldots, b_k)$ of A, respectively B with respect to $(g_1, \ldots, g_k)$. In our cash system, $\mathcal{P}$ knows at most one representation $(c_1, \ldots, c_k)$ of AB (we assume for the moment that he knows exactly one), and during the withdrawal phase he splits AB at random in two parts A, B (in $k+1$ parts in the general case of k-show cash). By Corollary 8, always $a_i + b_i = c_i \bmod q$. Suppose the protocol at the shop is such that $\mathcal{P}$ must release information about say $c_{j+1}, \ldots, c_k$ such that two executions of the protocol enable $\mathcal{V}$ to extract $c_{j+1}, \ldots, c_k$, and only one execution is needed to obtain $c_1, \ldots, c_j$ (for some j such that $o \leq j \leq k$). To this end, information about $c_1, \ldots, c_j$ is released in the clear (points) by $\mathcal{P}$, and about $c_{j+1}, \ldots, c_k$ in the form of lines.

The following exemplary protocol enables $\mathcal{P}$ to reveal points and lines about his representation of AB in such a way that $\mathcal{V}$ accepts if and only he is honest. Making use of the splitting A, B, this protocol then is as follows:

Step 1 $\mathcal{P}$ sends A, B, $\text{sign}(A, B)$ to $\mathcal{V}$ as well as the $2j$ responses $r_{1A} = a_1$, $r_{1B} = b_1$, $\ldots$, $r_{jA} = a_j$, and $r_{jB} = b_j$.

Step 2 $\mathcal{V}$ sends a challenge $c \in \mathbb{Z}_q \setminus \{1\}$ to $\mathcal{P}$.

Step 3 If $c \neq 1$, $\mathcal{P}$ computes the $k - j$ responses $r_{j+1} = a_{j+1} + cb_{j+1} \bmod q$, $\ldots$, $r_k = a_k + cb_k \bmod q$, and sends them to $\mathcal{V}$.

As in method 2, $\mathcal{V}$ accepts if and only if

$$(A/\prod_{i=1}^{j} g_i^{r_{iA}}) \cdot (B/\prod_{i=1}^{j} g_i^{r_{iB}})^c = \prod_{i=j+1}^{k} g_i^{r_i}.$$

Again, suppose at least one of the $k + j$ responses of $\mathcal{P}$ is incorrectly formed ($\mathcal{P}$ cheated). Rewriting the verification relation gives us

$$AB^c = g_1^{r_{1A}+cr_{1B}} \cdots g_j^{r_{jA}+cr_{jB}} g_{j+1}^{r_{j+1}} \cdots g_k^{r_k}.$$

That is,

$$(a_1 + cb_1 - (r_{1A} + cr_{1B}), \ldots, a_j + cb_j - (r_{jA} + cr_{jB}), a_{j+1} + cb_{j+1} - r_{j+1}, \ldots, a_k + cb_k - r_k)$$

is a representation of 1 with respect to $(g_1, \ldots, g_k)$. If one of the responses is incorrect, this representation is nontrivial, and we have a contradiction (Proposition 7). For a similar reason, if $\mathcal{P}$ starts out with knowing no representation at all of AB, he cannot have $\mathcal{V}$ accept in the protocol.

Therefore, $\mathcal{P}$ can only have $\mathcal{V}$ accept in this protocol if he knows a representation of AB, and is honest. The requirements are met: from one execution of the protocol, $\mathcal{V}$ can compute $c_i = r_{iA} + r_{iB} \bmod q$ for $1 \leq i \leq j$, and gets no information about the rest of the c_i's (strictly speaking, there must be a power of a dummy generator in AB: we discuss this further in

Section 13). From two executions of the protocol (with different challenges), $\mathcal{V}$ can also compute c_i for $j < i \leq k$.

It is easy to see that, for the correctness of the protocol, it does not matter whether the points are revealed in Step 1 or in Step 3. Observe that the reason why $c \neq 1$ is that in that case c_i can be computed for $j < i \leq k$ from just one execution of the protocol.

In general, we can apply this idea to polynomials of degree k, using a splitting in $k+1$ parts instead of in 2 parts. We will state and prove the general result in a formal way in the full paper.

11 The basic cash system

We now show the basic form of our cash system, i.e. in this system we only have coins (of various denominations). We describe this here using a user identity of the form (u_1, u_2) (and the bank knows $g_1^{u_1} g_2^{u_2}$), which also has the feature that framing of the bank of a user is prevented unconditionally. If one is content with a system in which framing is only computationally prevented, we can get an even more efficient scheme by having the identity of the user being simply of the form u (and the bank knows g^u).

11.1 The set-up of the system

The set-up of the system consists of the bank generating at random the following five (distinct) elements:

1. A generator-tuple (g, h). This is the public key of the bank, which it uses to make signatures with in the withdrawal protocol. The index $x = \log_g h$ is obviously kept secret.

2. A generator-tuple (g_1, g_2) which is used throughout the system.

3. A so-called dummy generator d (unequal to g_1 and g_2), which is to ensure that the identity of honest users cannot be computed from an execution of the payment protocol (independent of computing power).

For now, we assume that there is only one denomination (coin) in the system, and so we do not yet have to deal with how to encode different denominations. We discuss how to do this in the end of this section.

The safety of our scheme depends on no one being able to express any of g, h, g_1, g_2, d as a combination of powers of the other elements. Since this is at least as difficult as the problem of finding a nontrivial representation of 1 with respect to the generator-tuple (g, h, g_1, g_2, d), by Proposition 7 this is guaranteed to be infeasible (assuming the Discrete Log assumption). We furthermore remark that not even the bank need know any discrete logarithms except for x. As we discussed earlier in this paper, we can make use of this in various stages of the proof of security (but not privacy) of our cash system.

When a user $\mathcal{U}$ opens an account at the bank, he generates at random (possibly in cooperation with $\mathcal{B}$, using the protocol of 10.1) numbers $u_1, u_2 \in \mathbb{Z}_q$, and computes $I = g_1^{u_1} g_2^{u_2}$. The bank, not knowing (u_1, u_2), stores I together with the user's real identity and account number. In case the user double-spends the same cash, the bank will be able to find out (u_1, u_2). By computing $I = g_1^{u_1} g_2^{u_2}$, he can then determine his account and hence his identity. Therefore, we will speak more loosely of (u_1, u_2) or $g_1^{u_1} g_2^{u_2}$ of a user as being "his identity".

11.2 The withdrawal protocol

When a user $\mathcal{U}$ wants to withdraw a coin from his account (corresponding to say $I = g_1^{u_1} g_2^{u_2}$), he first convinces $\mathcal{B}$ that the money will indeed be withdrawn from his own account. To this end, $\mathcal{U}$ proves to $\mathcal{B}$ knowledge of a representation, using for example the scheme discussed in Section 8. We here observe that the user might show it in the clear, although this might not be smart because the bank can then frame him or there might be eavesdroppers who can then misrepresent themselves. If $\mathcal{B}$ accepts $\mathcal{U}$'s proof then (for each coin that $\mathcal{U}$ wishes to withdraw) the actual withdrawal protocol is executed (see Figure 5).

Step 1 $\mathcal{B}$ subtracts the appropriate amount of money (which we assumed for the moment to be fixed) from the account and forms the number $m = Id$ (note that $\mathcal{U}$ can compute this number himself, so there is no need for $\mathcal{B}$ to transmit it to $\mathcal{U}$). $\mathcal{B}$ sends m^x, g^w and m^w to the user, with w randomly chosen from $\mathbb{Z}_q$.

Step 2 $\mathcal{U}$ generates random numbers $s \in_{\mathcal{R}} \mathbb{Z}_q^*$, $u, v \in_{\mathcal{R}} \mathbb{Z}_q$, and transforms m, using s, to $m' (= m^s = g_1^{u_1 s} g_2^{u_2 s} d^s)$. He now randomly splits each of the three exponents in two parts, i.e., determines splittings $u_1 s = x_1 + x_2 \bmod q$, $u_2 s = y_1 + y_2 \bmod q$ and $s = z_1 + z_2 \bmod q$, and then computes $A = g_1^{x_1} g_2^{y_1} d^{z_1}$. He then computes $B = m^s / A$, and also the other blinded terms (using u, v) for the hash-function (we call them, as before, z', a', b'). He then sends challenge $c = \mathcal{H}(m', z', a', b', A)/u \bmod q$ to $\mathcal{B}$.

Step 3 $\mathcal{B}$ sends the response back as in the blind signature scheme, and $\mathcal{U}$ transforms it to a corresponding response r'.

In Step 2, there is actually no need for the bank to send m^x to the user: in the set-up phase, the bank can also publish (g_1^x, g_2^x, d^x), so each user can compute m^x corresponding to their own identity (but by the Diffie-Hellman assumption not those of others if one is to end up with knowledge of a representation, although this is not important for the security of the system). This has the advantage that the user can do even more precomputing before the actual withdrawal protocol begins (see Section 15).

By Proposition 10, $\mathcal{U}$ ends up with a triple $(A, B, \mathrm{sign}(A, B) = (z', a', b', r'))$ that is unconditionally unlinkable to the withdrawal, and for which $AB = m'$. Note that if $\mathcal{U}$ followed the protocol, then he knows representations of m', A and B with respect to (g_1, g_2, d) (respectively $(u_1 s, u_2 s, s)$, (x_1, y_1, z_1), and (x_2, y_2, z_2)). By Corollary 8, the representations are always such that $u_1 s = x_1 + x_2 \bmod q$, $u_2 s = y_1 + y_2 \bmod q$ and $s = z_1 + z_2 \bmod q$ (otherwise the Discrete Log assumption was broken since, as we discussed, not even the bank needs to know a nontrivial representation of 1 with respect to (g_1, g_2, d)).

11.3 The payment.

When $\mathcal{U}$ wants to spend his coin at a shop $\mathcal{S}$, the following protocol is executed (see Figure 6):

$$
\begin{array}{lcr}
\mathcal{U} & & \mathcal{B} \\
(I = g_1^{u_1} g_2^{u_2}) & & (g, h = g^x) \\
 & \xleftrightarrow{\text{Proof of knowledge of } (u_1, u_2)} & \\
 & \vdots & \\
 & & w \in_{\mathcal{R}} \mathbb{Z}_q \\
 & & m \leftarrow Id \\
 & & z \leftarrow m^x \\
 & & a \leftarrow g^w \\
 & \xleftarrow{\quad z, a, b \quad} & b \leftarrow m^w \\
m \leftarrow g_1^{u_1} g_2^{u_2} d & & \\
s \in_{\mathcal{R}} \mathbb{Z}_q^* & & \\
m' \leftarrow m^s & & \\
z' \leftarrow z^s & & \\
\text{Split } m' \text{ in } A, B & & \\
u, v \in_{\mathcal{R}} \mathbb{Z}_q & & \\
a' \leftarrow a^u g^v & & \\
b' \leftarrow b^{su} (m')^v & & \\
c' \leftarrow \mathcal{H}(m', z', a', b', A) & & \\
c \leftarrow c'/u \bmod q & \xrightarrow{\quad c \quad} & \\
 & \xleftarrow{\quad r \quad} & r \leftarrow xc + w \bmod q \\
g^r \stackrel{?}{=} h^c a & & \\
m^r \stackrel{?}{=} z^c b & & \\
r' \leftarrow ru + v \bmod q & &
\end{array}
$$

FIGURE 5. Withdrawal protocol for coins

$$\begin{array}{lcr}
\mathcal{U} & & \mathcal{S} \\
 & \xrightarrow{\quad A, B, \mathrm{sign}(A,B) \quad} & \\
 & & AB \overset{?}{\neq} 1 \\
 & & \text{Verify } \mathrm{sign}(A,B) \\
 & \xleftarrow{\qquad\quad c \qquad\quad} & c \in \mathbb{Z}_q^* \\
r_1 \leftarrow x_1 + cx_2 \bmod q & & \\
r_2 \leftarrow y_1 + cy_2 \bmod q & & \\
r_3 \leftarrow z_1 + cz_2 \bmod q & \xrightarrow{\qquad r_1, r_2, r_3 \qquad} & \\
 & & g_1^{r_1} g_2^{r_2} d^{r_3} \overset{?}{=} AB^c
\end{array}$$

FIGURE 6. Payment protocol for coins

Step 1 $\mathcal{U}$ sends the triple

$$A = g_1^{x_1} g_2^{y_1} d^{z_1}, B = g_1^{x_2} g_2^{y_2} d^{z_2}, \mathrm{sign}(A,B) = (z', a', b', r')$$

to $\mathcal{S}$. (In case there are different denominations in the system, $\mathcal{U}$ also has to inform $\mathcal{S}$ of the amount of money the coin is supposedly worth.)

Step 2 $\mathcal{S}$ first verifies that $AB \neq 1$. Then $\mathcal{S}$ verifies the signature, by first computing $c' = \mathcal{H}(AB, z', a', b', A)$ and verifying with this c' the relations as in the basic blind signature protocol. If the verification holds, $\mathcal{S}$ sends a challenge $c \in \mathbb{Z}_q^*$ to $\mathcal{U}$.

Step 3 $\mathcal{U}$ responds with $r_1 = x_1 + cx_2 \bmod q$, $r_2 = y_1 + cy_2 \bmod q$, $r_3 = z_1 + cz_2 \bmod q$.

$\mathcal{S}$ verifies whether $g_1^{r_1} g_2^{r_2} d^{r_3} = AB^c$, and, if so, accepts. Note that if $AB = 1$, then s was chosen to be zero, which is not allowed since u_1, u_2 will then not be revealed when the user double-spends.

11.4 The deposit.

After some units of time, all the shops send their gathered information of payments to $\mathcal{B}$ (see Figure 7), who will now immediately discover any double-spent coins, since the same A, B, $\mathrm{sign}(A,B)$ have been used. From the responses to the challenges, $\mathcal{B}$ can compute $x_1, x_2, y_1, y_2, z_1, z_2$, and hence the identifying information of the user who double-spent as:

$$u_1 = (x_1 + x_2)(z_1 + z_2)^{-1} \bmod q \text{ and } u_2 = (y_1 + y_2)(z_1 + z_2)^{-1} \bmod q.$$

Hence $\mathcal{B}$ can compute $I = g_1^{u_1} g_2^{u_2}$, and search its account-list to identify the double-spender.

For later comparison with the way the bank identifies a double-spender in the wallet-with-observer setting (Section 16), we here derive the precise formulas that the bank needs to

$$\begin{array}{ccc} \mathcal{S} & & \mathcal{B} \\ & \xrightarrow{A,\, B,\, \mathrm{sign}(A,B),\, c,\, r_1,\, r_2,\, r_3} & \\ & & \text{verification as in the} \\ & & \text{payment protocol, and} \\ & & \text{check for double-spending} \end{array}$$

FIGURE 7. Deposit protocol for coins

apply to extract u_1, u_2. If the bank detects that information A, B, $\mathrm{sign}(A, B)$ has been spent more than once, it then has at its disposal two sets of responses (r_1, r_2, r_3) and (r'_1, r'_2, r'_3), corresponding to two different challenges c and c'. By the verification relations, the following equations must hold:

$$g_1^{r_1} g_2^{r_2} g_3^{r_3} = AB^c,$$
$$g_1^{r'_1} g_2^{r'_2} g_3^{r'_3} = AB^{c'}.$$

From this, we derive that

$$A = g_1^{(c'r_1 - cr'_1)/(c'-c)} g_2^{(c'r_2 - cr'_2)/(c'-c)} g_3^{(c'r_3 - cr'_3)/(c'-c)},$$
$$B = g_1^{(r_1 - r'_1)/(c-c')} g_2^{(r_2 - r'_2)/(c-c')} g_3^{(c'r_3 - cr'_3)/(c'-c)}.$$

We know that

$$A = g_1^{x_1} g_2^{y_1} g_3^{z_1},$$
$$B = g_1^{x_2} g_2^{y_2} g_3^{z_2}.$$

Since the system cannot have come up with a non-trivial representation of 1 with respect to (g_1, g_2, g_3), it must be that

$$\begin{aligned} (c'r_1 - cr'_1)/(c'-c) &= x_1 \bmod q, \\ (c'r_2 - cr'_2)/(c'-c) &= y_1 \bmod q, \\ (c'r_3 - cr'_3)/(c'-c) &= z_1 \bmod q, \\ \\ (r_1 - r'_1)/(c-c') &= x_2 \bmod q, \\ (r_2 - r'_2)/(c-c') &= y_2 \bmod q, \\ (r_3 - r'_3)/(c-c') &= z_2 \bmod q. \end{aligned}$$

This in turn implies that the bank can compute

$$\begin{aligned} u_1 s &= x_1 + x_2 = (c'r_1 - cr'_1)/(c'-c) + (r_1 - r'_1)/(c-c') \bmod q, \\ u_2 s &= y_1 + y_2 = (c'r_2 - cr'_2)/(c'-c) + (r_2 - r'_2)/(c-c') \bmod q, \\ s &= z_1 + z_2 = (c'r_3 - cr'_3)/(c'-c) + (r_3 - r'_3)/(c-c') \bmod q, \end{aligned}$$

and so

$$u_1 = (\, r_1(c'+1) - r'_1(c+1))\,)/(\, r_3(c'-1) - r'_3(c+1)\,) \bmod q,$$
$$u_2 = (\, r_2(c'+1) - r'_2(c+1))\,)/(\, r_3(c'-1) - r'_3(c+1)\,) \bmod q.$$

11.5 Remarks concerning denominations, the shop's challenge, framing and k-show coins

Various denominations There are various ways in which we can have our system consist of coins with several distinct denominations. The most obvious of these is (like in cash systems based on RSA, where usually the root that the bank extracts denotes the denomination) to have the bank use a different public key for each denomination. That is, if there are to be k distinct coins, the bank publishes $(g, h_1), \ldots, (g, h_k)$ as its public keys, and $x_1, \ldots, x_k$ (with $x_i = \log_{g_i} h_i$) are its corresponding secret keys.

A more efficient (and elegant) way to encode denominations, which makes use of the structure in the representation problem, is to have k dummy generators d_1, d_k published by the bank (in this context, it would make more sense to speak of denomination generators than of dummy generators). Each of these generators is used to denote some fixed amount of money (i.e. a coin). This encoding has the advantage that one can have d_i denote for example $\$2^{i-1}$, and hence one can then use all denominations from \$1 up to $\$2^k$, multiplying suitable generators according to the binary expansion of the denomination. For example, \$11 would be represented by $g_1 g_2 g_4$. That is, with k generators, $2^k - 1$ different amounts can be represented, without this affecting the efficiency of the protocols for withdrawal, payment, and deposit.

We observe here that, when this latter encoding of denominations is used there is no way that a user can withdraw say \$1 (corresponding to d_1), and spent it in a shop for say \$6 (corresponding to $d_2 d_3$) for a reason discussed already before in a different context: assuming that in the withdrawal protocol the user ends up with a number m (plus $\text{sign}(m)$) of which he knows a representation with respect to (g_1, g_2, d_1), if he succeeds in having the shop accept this information for \$6, he must necessarily know a representation of m with respect to $(g_1, g_2, d_2 d_3)$. Hence, the user knows a non-trivial representation of 1 with respect to $(g_1, g_2, d_1, d_2 d_3)$. However, not even the bank needs to know this, and can be assumed to be (since the security for the bank needs only be proven for an honest bank) polynomially bounded. This would imply that the system (viewed as a probabilistic polynomial-time algorithm whose random choices simulate the actions of its participants) has come up with a nontrivial representation of 1 whereas it only got inputs $(g, h, g_1, g_2, d_1, \ldots, d_k)$. Note that the only situation in which this argument, which clearly holds in general, does not give us a contradiction is if the user in the withdrawal protocol ends up with an m of which he knows indeed a representation with respect to $(g_1, g_2, d_2 d_3)$. We discuss this possibility further in Section 12.

The choice of the shop's challenge The challenge c from $\mathcal{S}$ in the payment protocol can (and probably should be in a realistic implementation) be computed as a one-way hash-function of several data (the first transmissions, the time of transaction etc.). In particular, the challenge should be unique to each shop, hence the value of the above function should be concatenated with a string that was generated by the bank, and which is unique for each shop. Therefore, in principle the entire payment protocol can be condensed to a single move. We refer the reader to [8] for a further discussion of this matter. Observe that even if c is generated (to some extent) at random, and hence the responses must be computed in real time, this still is extremely efficient (two multiplications and additions). We believe this constitutes as fast a payment as one would like to have from the point of view of the user.

Framing If the bank falsely accuses a user of having double-spent a coin, the user can falsify this before a judge. To this end, the bank has to tell the representation (say (u_1, u_2)) it found. If this representation is not that of the user (which is the case with overwhelming probability if the user was honest), the user now knows a non-trivial representation of 1 with respect to (g_1, g_2) (or $\log_{g_1} g_2$, for that matter). Since he could not have come up with this by himself, it proves that the bank tried to frame the user. We note that it would in fact be very unwise for the bank to try to frame a user, since the user could prefer not to deny this. Since this user would then know $\log_{g_1} g_2$, he can from this moment on double-spend to his heart's content, without ever being caught.

In our basic cash system, the identity of the user is encoded as $g_1^{u_1} g_2^{u_2}$, which is to make sure that framing attempts of the bank have negligible probability of success, regardless of computing power. We can also encode the user-identity more simply as g^u. The entire system then becomes even a little more efficient (only two responses are needed in the payment protocol, and the verification relations are easier to compute). However, as we discussed before, the bank usually will choose the security parameters, so it may be better to prevent framing unconditionally rather than it being merely infeasible. This remark also applies to all extensions we will discuss.

Multi-show cash Using the general protocol described in subsection 10.5, the protococols can be straightforwardly adapted in such a way that the user can spend the same coin say k times. If he spends it for the $(k+1)$-th time ("double-spending"!), his identity can be extracted by the bank. To this end, the user at the withdrawal splits m' into $k+1$ parts, and in the payment phase gives responses that are polynomials of degree k in the challenge. Clearly, at withdrawal time the bank then subtracts k times the value of the coin from the account of the user, in the payment phase the user also has to inform the shop of the number of times he is allowed to spend the coin, and $\mathcal{H}(\cdot)$ must be a has-function with $k+4$ inputs. Multi-show cash can be advantageous in certain circumstances, since the storage requirement of the user much less then if he would store k copies of the coin. Moreover, contrary to checks, no refund protocol is needed, and the withdrawal protocol remains about as efficient as it was. When tre user pays with a multi-show coin in one transaction, there is no disadvantage with respect to his privacy at all over paying with a check or many one-show coins.

12 The correctness of the basic cash system

In this section we sketch the correctness of our cash system. a detailed proof of the contents of this section will be in the full paper. First, we remark that actually everything has already been proven about the privacy of the honest users, and almost everything concerning the security (the privacy of honest users is unconditionally guaranteed due to Propositions 10 and 13). It remains to show that in the withdrawal protocol, the user cannot obtain signatures on numbers of which he knows a representation for which the structure in the representation (in which we encode his identity) no longer remains.

In order to prove this, we make use of Proposition 12 in combination with some extra restrictions that must hold. The first restriction is that at the start of the withdrawal protocol, a number m is used of which the user knows exactly one representation $(u_1, u_2, 1)$ with respect to (g_1, g_2, d). This is true since the user proves knowledge of a representation (u_1, u_2) of (g_1, g_2) when he wants to get access to his account, and not even the bank need know any nontrivial representation of 1 with respect to (g_1, g_2, d). The second restriction is that if the user wants to have any chance of having the shop accept in the payment protocol, he must know at least one representation of m' with respect to (g_1, g_2, d). Again, since not even the bank need know any nontrivial representation of 1 with respect to (g_1, g_2, d), the best that the user can do is to end up knowing exactly one representation of m'. As we discussed, the user will not be identified after double-spending, in case he can obtain a representation of m' that is not a multiple of the representation he knows of m. That is, during the blinding, he then has to multiply in e.g. a factor g_1^r in $(g_1^{u_1} g_2^{u_2} d)^s$. However, it is easy to prove that he does not know a representation of g_1 with respect to (m, g), since he then would know a non-trivial representation of 1. As in the proof of Proposition 12, we hence have that the only blinding manipulations that the user can perform are such that his representation of m' is a multiple of his representation of m, and this holds independent of the information he may have gathered from polynomially many transcripts of previous executions of the withdrawal protocol.

Furthermore, we observe that it is infeasible to forge pairs m, $\text{sign}(m)$. This follows directly from the choice of the hash-function (see [15]). We remark that even if the user could obtain such a pair, he can only pay with it if he knows a representation of m with respect to (g_1, g_2, d). Hence, his task to forge cash is even harder: not only must he come up with a pair m, $\text{sign}(m)$, but he must also know a representation of m.

Finally, we outline the proof of the infeasibility of forging transcripts of valid executions of the payment protocol. In fact, this is an almost immediate consequence of the fact that the challenge must be chosen as a one-way hash function of the information sent in Step 1. Hence, well-known arguments show that forgeability is infeasible (since the protocol is constructed in fact by the Fiat-Shamir technique to transform minimum-knowledge protocols to signature schemes). Again, we observe that the task of the forger is actually even harder: even if he could find A, B such that he obtains a valid transcript of the protocol (for example by working "backwards" or applying a meet-in-the-middle attack), he must also determine $\text{sign}(A, B)$, which is infeasible.

13 Extension to checks

In this section, many symbols are needed to denote the indices and responses and we consequently even run out of letters of the alphabet. We therefore remark here that it might be desirable to use a vector notation, such as $\mathbf{g}^{\mathbf{a}}$ instead of $\prod_{i=1}^{k} g_i^{a_i}$. In the full paper, we will introduce such a notation.

The basic payment protocol is quite efficient, and does not require that much storage space if a user pays at a shop with a bunch of coins. However, although introducing checks in the system has the disadvantage that an extra protocol for refund is necessary, it can still be more efficient when amounts that can only be paid with many coins are involved.

We first sketch the basic idea of how to extend the basic system to one that includes checks. Suppose we have a generator-tuple $(f_1, \ldots, f_k)$ of length k, with f_i representing some amount of money, say $\$2^{i-1}$. As we discussed in Section 11, any amount up to $\$2^k - 1$ can then be represented by multiplying the appropriate generators according to the binary expansion of the particular amount. Due to the vector addition chain techniques, computations involving such amounts are still efficient. During the withdrawal phase, a user who intends to withdraw a check that can be spent up to \$$2^k - 1$, generates at random an index-tuple $(a_1, \ldots, a_k)$ of length k, and computes $m = \prod_{i=1}^{k} f_i^{a_i}$. The a_i's are called refund terms. Neglecting all privacy issues for the moment, suppose the user has this number signed by the bank, and the bank stores this check m with the users account. At the shop, when the user wants to spend some amount of money (say \$7), he reveals not only $(m, \mathrm{sign}(m))$, but also (a_1, a_2, a_3), and the shop makes sure that he receives a signed proof of correctness of this information. At deposit phase, it sends this information to the bank, who (after verifying its correctness) stores (a_1, a_2, a_3) on a special refund-list. When the user wants to get a refund for the unspent part of the check m (corresponding to $f_4, \ldots, f_k$), he sends $a_4, \ldots, a_k$ to the bank. The bank verifies whether m is indeed stored with the user's account, and also that $a_4, \ldots, a_k$ are not already on its refund-list. To ensure that the user honestly sent the a_i's, he has to prove to the bank knowledge of a representation of $m / \prod_{i=4}^{k} f_i^{a_i}$ with respect to (f_1, f_2, f_3). If the bank accepts, it refunds the user the appropriate amount of money, erases m from the account of the user, and stores $(a_4, \ldots, a_k)$ on the refund-list.

Of course, since the check is unblinded, there is no privacy yet. Before we discuss how to achieve this, we remark that the bank can, after the refund, link the refund to numbers on the refund-list in the following way: for all tuples on the refund-list, the bank can simply find out which is a representation of $m / \prod_{i=4}^{k} f_i^{a_i}$ with respect to (f_1, f_2, f_3). Hence, an extra dummy generator d is needed, i.e. m is of the form $(\prod_{i=1}^{k} f_i^{a_i})\, d^{a_{k+1}}$. The index a_{k+1} also is a random choice of the user, and he never reveals it. This prevents the kind of linking just described, since for each tuple on the refund-list there is a number a_{k+1} such that this tuple is a representation with respect to $m / (\prod_{i=4}^{k} f_i^{a_i}\ d^{a_k+1})$.

We now incorporate anonymity. Since a user can cheat in this scheme (without being identified after the fact) by simply spending the same parts of the check over and over again in case of anonymity, the actual check system is somewhat more complicated. First of all, to ensure that the user remains anonymous, we have a check consist of $2k$ terms, $(e_1, f_1), \ldots, (e_k, f_k)$. The pair (e_i, f_i) replaces the f_i in the above discussion. The denomination part now becomes the product of powers of these $2k$ elements, and the quotients of the exponents of e_i, f_i will perform the role of the a_i above. The reason for this is the same as in the basic cash system, namely to make sure that during the blinding the refund terms are not altered. Secondly, to be able to identify a user who spends (the same parts of) a check more than once, we in addition build in the double-spending property as in the basic cash system. A check in our system hence looks like the following product of powers of $2k + 4$ different generators (chosen by the bank):

$$\overbrace{(\prod_{i=1}^{2} g_i^{u_i}) \cdot d_1^{z_1}}^{\text{identity-part}} \cdot \overbrace{(\prod_{i=1}^{k} e_i^{a_i} f_i^{b_i}) \cdot d_2^{a_{k+1}}}^{\text{denomination-part}} .$$

As in the basic cash system, a user is known to the bank as $I = g_1^{u_1} g_2^{u_2}$ (or, as in the basic system, as $I = g_1^{u}$ if it is sufficient that framing of the bank is infeasible rather than having negligible success probability independent of computing power). The generators d_1, d_2 are dummies, the purpose of d_1 being the same as that of d in the cash system (i.e. to make sure the identity of the user is still hidden unconditionally after spending the check once and having requested refund), and that of g_2 being to prevent linking by the bank after the refund phase.

13.1 The withdrawal protocol

After the user has identified himself to the bank as in the basic cash system, the following withdrawal protocol is performed (see Figure 8):

Step 1 $\mathcal{U}$ generates at random $2k+1$ distinct non-zero random numbers, a_i, b_i (with $1 \leq i \leq k$) and a_{k+1}. He then computes $m_1 = (\prod_{i=1}^{k} e_i^{a_i} f_i^{b_i}) d_2^{a_{k+1}}$, and sends this to $\mathcal{B}$.

Step 2 $\mathcal{B}$ subtracts the maximum amount for which the check can be spend from the $\mathcal{U}$'s account, and lists m_1 with $\mathcal{U}$'s account. With $m_2 = Id_1$, $\mathcal{B}$ then computes $m = m_1 m_2$, and sends m^x, g^w and m^w to the user, with w randomly chosen from $\mathbb{Z}_q$.

Step 3 $\mathcal{U}$ generates random numbers $s \in_{\mathcal{R}} \mathbb{Z}_q^*$, $u, v \in_{\mathcal{R}} \mathbb{Z}_q$, and transforms m, using s, to m'. Note that

$$m' = m^s = (g_1^{u_1 s} g_2^{u_2 s} d_1^s)\,(e_1^{a_1 s} f_1^{b1 s} \cdots e_k^{a_k s} f_k^{b_k s} d_2^{a_{k+1} s}).$$

He now randomly splits each of the $2k+4$ exponents into two parts, i.e., determines splittings

$$\begin{aligned}
u_1 s &= x_1 + x_2 \bmod q \\
u_2 s &= y_1 + y_2 \bmod q \\
s &= z_1 + z_2 \bmod q \\
\\
a_1 s &= \alpha_{1A} + \alpha_{1B} \bmod q \\
b_1 s &= \beta_{1A} + \beta_{1B} \bmod q \\
&\vdots \\
a_k s &= \alpha_{kA} + \alpha_{kB} \bmod q \\
b_k s &= \beta k A + \beta_{kB} \bmod q \\
a_{k+1} s &= \alpha_A + \alpha_B \bmod q
\end{aligned}$$

and computes

$$A = g_1^{x_1} g_2^{y_1} d_1^{z_1}\, e_1^{\alpha_{1A}} f_1^{\beta_{1A}} \cdots e_k^{\alpha_{kA}} f_k^{\beta_{kA}} d_2^{\alpha_A}.$$

$\mathcal{U}$ then computes $B = m'/A$, and also the other blinded terms (using u, v) for the hash-function (we call them, as before, z', a', b'). He then sends challenge

$$c = \mathcal{H}(m', z', a', b', A)/u \bmod q$$

to $\mathcal{B}$.

Step 4 $\mathcal{B}$ sends the response back as in the blind signature scheme, and $\mathcal{U}$ transforms it to a corresponding response r'.

Again, as in the basic cash system, there is actually no need for the bank to send m^x to the user.

We note that for checks the public key of the bank must be different from that used to sign coins. If this were not the case, a user could ask for a check in the withdrawal, send $m_1 = g_2^t$ for some random t to the bank, and receive a signature on m^r from the bank, with $m = g_1^{u_1} g_2^{u_2+t} d_1$. Hence, if he could use this information as a coin, he could double-spend it without his identity falling out. Another way to prevent this than using another public key is for the bank to require of the user that (after Step 1) he proves knowledge of m_1 with respect to the generator-tuple $(e_1, f_1, \ldots, e_k.f_k, d_2)$, but this is less efficient and not necessary in view of the above.

$\mathcal{U}$		$\mathcal{B}$
$(g_1^{u_1} g_2^{u_2})$		$(g, h = g^x)$
	$\xleftrightarrow{(\text{Proof of knowledge of } u_1, u_2)}$	
	$\vdots$	
$a_1, \ldots, a_{k+1} \in_{\mathcal{R}} \mathbb{Z}_q^*$ $b_1, \ldots, b_k \in_{\mathcal{R}} \mathbb{Z}_q^*$ (all different)		
$m_1 \leftarrow \prod_{i=1}^{k} e_i^{a_i} f_i^{b_i} \cdot d_2^{a_{k+1}}$	$\xrightarrow{m_1}$	
		(store m_1) $w \in_{\mathcal{R}} \mathbb{Z}_q$ $m \leftarrow Id_1 m_1$ $z \leftarrow m^x$ $a \leftarrow g^w$
	$\xleftarrow{z, a, b}$	$b \leftarrow m^w$
$m \leftarrow g_1^{u_1} g_2^{u_2} d_1 m_1$ $s \in_{\mathcal{R}} \mathbb{Z}_q^*$ $m' \leftarrow m^s$ $z' \leftarrow z^s$ Split m' in A, B $u, v \in_{\mathcal{R}} \mathbb{Z}_q$ $a' \leftarrow a^u g^v$ $b' \leftarrow b^{su} (m')^v$ $c' \leftarrow \mathcal{H}(m', z', a', b', A)$		
$c \leftarrow c'/u \bmod q$	$\xrightarrow{c}$	
	$\xleftarrow{r}$	$r \leftarrow xc + w \bmod q$
$g^r \stackrel{?}{=} h^c a$ $m^r \stackrel{?}{=} z^c b$ $r' \leftarrow ru + v \bmod q$		

FIGURE 8. Withdrawal protocol for checks

13.2 The payment protocol

When $\mathcal{U}$ wants to spend his check at a shop $\mathcal{S}$, the following protocol is executed (see Figure 9):

Step 1 $\mathcal{U}$ sends the triple $A, B, \text{sign}(A, B)$ to $\mathcal{S}$. $\mathcal{U}$ also informs $\mathcal{S}$ of the amount of money for which he wishes to spend this check. Without loss of generality, we assume he wishes to spend an amount corresponding to $(e_1, f_1), \ldots, (e_j, f_j)$ (i.e. $\$2^j - 1$ if the denominations are as in the example), with $1 \leq j \leq k$.

Step 2 $\mathcal{S}$ first verifies that $AB \neq 1$. Then $\mathcal{S}$ verifies the signature as before and, if the verification holds, sends a challenge $c \in \mathbb{Z}_q^* \setminus \{1\}$ to $\mathcal{U}$.

Step 3 $\mathcal{U}$ verifies that $c \neq 1$, and if this holds he computes (and transmits to $\mathcal{V}$) the appropriate responses:

$$\begin{array}{rcl}
\text{For } i = 1 \text{ to } j \text{ (points):} & & \\
r_i = (r_{i1},\ r_{i2},\ r_{i3},\ r_{i4}) & \leftarrow & (\alpha_{iA},\ \alpha_{iB},\ \beta_{iA},\ \beta_{iB}) \\
& & \\
\text{For } i = j+1 \text{ to } k \text{ (lines):} & & \\
r_i = (r_{i1},\ r_{i2}) & \leftarrow & (\alpha_{iA} + c\alpha_{iB} \bmod q,\ \beta_{iA} + c\beta_{iB} \bmod q) \\
& & \\
r_{k+1} = (r_{(k+1)1}, r_{(k+1)2}, r_{(k+1)3}) & \leftarrow & (x_1 + cx_2 \bmod q,\ y_2 + cy_2 \bmod q,\ z_1 + cz_2 \bmod q) \\
r_{k+2} & \leftarrow & \alpha_A + c\alpha_B \bmod q.
\end{array}$$

That is, for each part he wishes to spend, he reveals the corresponding numbers (points), and for the identity-part he reveals lines (corresponding to r_{k+1}. The other lines (corresponding to $r_{j+1}, \ldots, r_k$ and r_{k+2}) are, as before, in fact merely to ensure the self-certifiability property.

The points might as well be sent before the challenge, in Step 1.

$\mathcal{S}$ verifies this information as follows (see Subsection 10.5):

$$r_{13} + r_{14} \stackrel{?}{\neq} 0,$$
$$\vdots$$
$$r_{j3} + r_{j4} \stackrel{?}{\neq} 0,$$

$$AB \stackrel{?}{\neq} 1,$$

$$A\,B^c \stackrel{?}{=} \prod_{i=1}^{j} e_i^{r_{i1}} f_i^{r_{i3}} \cdot (\prod_{i=1}^{j} e_i^{r_{i2}} f_i^{r_{i4}})^c \cdot \prod_{i=j+1}^{k} e_i^{r_{i1}} f_i^{r_{i2}} \cdot \prod_{i=1}^{2} g_i^{r_{(k+1)i}} \cdot d_1^{r_{(k+1)3}} d_2^{r_{k+2}},$$

and accepts if and only if these verifications hold.

13.3 The deposit protocol

After some units of time, $\mathcal{S}$ sends all the information of the payment to $\mathcal{B}$ (see Figure 10). This consists of

$$A, B, \text{sign}(A, B), r_1, \ldots, r_{k+2}.$$

$$
\begin{array}{lcr}
\mathcal{U} & & \mathcal{S} \\
 & \xrightarrow{A,\,B,\,\mathrm{sign}(A,B)} & \\
 & & AB \overset{?}{\neq} 1 \\
 & & \text{Verify sign}(A,B) \\
 & \xleftarrow{\quad c \quad} & c \in \mathbb{Z}_q^* \setminus \{1\} \\
c \overset{?}{\neq} 1 & & \\
\text{For } i = 1 \text{ to } j\text{:} & & \\
r_i \leftarrow (\alpha_{iA},\ \alpha_{iB},\ \beta_{iA},\ \beta_{iB}) & & \\
 & & \\
\text{For } i = j+1 \text{ to } k\text{:} & & \\
r_i \leftarrow (\alpha_{iA} + c\alpha_{iB} \bmod q, & & \\
\qquad \beta_{iA} + c\beta_{iB} \bmod q) & & \\
 & & \\
r_{k+1} \leftarrow (x_1 + cx_2 \bmod q, & & \\
\qquad y_2 + cy_2 \bmod q, & & \\
\qquad z_1 + cz_2 \bmod q) & & \\
r_{k+2} \leftarrow \alpha_A + c\alpha_B \bmod q & \xrightarrow{r_1,\ldots,r_{k+2}} & \\
 & & \text{(Verify)}
\end{array}
$$

FIGURE 9. Payment protocol for checks

$\mathcal{S}$ $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ $\mathcal{B}$

$$\xrightarrow{A, B, \mathrm{sign}(A, B), c, r_1, \ldots, r_{k+2}}$$

(Verification as in the
payment protocol, and
check for double-spending)

(Store $(r_{i1} + r_{i2})(r_{i3} + r_{i4})^{-1} \bmod q$
on refund list, for $1 \leq i \leq j$)

FIGURE 10. Deposit protocol for checks

The bank verifies it, applying precisely the same verifications as $\mathcal{S}$ did after Steps 1 and 3 of the payment. If the verifications hold then $\mathcal{B}$ stores, for $i = 1$ to j, the following terms on the refund list:

$$(r_{i1} + r_{i2})(r_{i3} + r_{i4})^{-1} \bmod q.$$

Observe that if the check has been double-spent, $\mathcal{B}$ detects this and distills (from two responses r_{k+1} and r'_{k+1}) the identity of the double-spender as in the basic cash system.

13.4 The refund protocol

When the user wants a refund for the amount of the check which he did not spent, the user first informs the bank of his account number, and of m_1. The bank verifies that this m_1 is listed with the user's account and, if so, the following protocol is performed (see Figure 11):

Step 1 $\mathcal{U}$ sends a_i, b_i, for $j < i \leq k$, to $\mathcal{B}$ (i.e. the points corresponding to the part of the check which is to be refunded).

Step 2 $\mathcal{B}$ verifies that for all $j < i \leq k$, $b_i \neq 0$. If this holds, $\mathcal{B}$ verifies that the quotients $a_i/b_i \bmod q$ that $\mathcal{U}$ sent to him in Step 1 are not already on the refund-list. If this is also the case, $\mathcal{U}$ proves to $\mathcal{B}$ knowledge of a representation of $m_1/(\prod_{i=j+1}^{k} e_i^{a_i} f_i^{b_i})$ with respect to $(e_1, f_1, \ldots, e_j, f_j, d_2)$. In Figure 11 we explicitly use the proof of knowledge of Section 8 for this.

Step 3 If $\mathcal{B}$ accepts, he refunds the appropriate amount of money to $\mathcal{U}$, and puts $a_i/b_i \bmod q$, for $j < i \leq k$, on the refund list. He then erases m_1 from $\mathcal{U}$'s account.

13.5 The security of the check extension

It is easy to prove that the untraceability of the user is guaranteed unconditionally, i.e. that the lines for the unopened denomination parts do not reveal any information that can later be used to link payments to refunds. The rest of the proof of correctness follows directly from

$\mathcal{U}$		$\mathcal{B}$
		Is m_1 listed with account?
	$\vdots$	
For $i = 1$ to $j + 1$: $v_i \in_{\mathcal{R}} \mathbb{Z}_q$		
For $i = 1$ to j: $w_i \in_{\mathcal{R}} \mathbb{Z}_q$		
$z \leftarrow (\prod_{i=1}^{j} e_i^{v_i} f_i^{w_i}) \cdot d_2^{v_{j+1}}$	$\xrightarrow{z, a_{j+1}, b_{j+1}, \ldots, a_k, b_k}$	
		For $j < i \leq k$: $b_i \stackrel{?}{\neq} 0$ a_i/b_i not on refund-list?
	$\xleftarrow{c}$	$c \in \mathbb{Z}_q$
For $i = 1$ to $j + 1$ do: $r_{i1} \leftarrow v_i + ca_i \bmod q$		
For $i = 1$ to j do: $r_{i2} \leftarrow w_i + cb_i \bmod q$	$\xrightarrow{r_{11}, r_{12}, \ldots, r_{j2}, r_{(j+1)1}}$	
		$z \cdot (m / \prod_{i=j+1}^{k} e_i^{a_i} f_i^{b_i})^c$ $\stackrel{?}{=} \prod_{i=1}^{j} e_i^{r_{i1}} f_i^{r_{i2}} \cdot d_2^{r_{(j+1)1}}$ erase m_1 from account of user for $1 \leq i \leq j$, store $a_i/b_i \bmod q$

FIGURE 11. Refund protocol

the proof of correctness of the basic cash system. Note that the basic cash system is in fact a special case of the check system, in that we just leave out the denomination part of the check. We will give a detailed proof of correctness in the full paper.

In our check extension, there still is the problem of what one might call linking by complementary amounts. This means that the bank can obtain some information from the amount for which a refund is requested, since on its refund-list complementary amounts appear. Hence, the bank can exclude many of the tuples on the refund-list, since they do not form the complementary amount to that requested for refund. This can be prevented to a great extent by having the user request for refund in batches (see [1]). We show how to do this in the full paper.

14 Extension to divisibility

We briefly sketch how to extend the system to divisibility here. The details will be worked out in the full paper. The underlying principle is actually the same as that used for checks. A divisible amount of money can be viewed as a check, the unspent parts of which can be spent until the total amount is reached. Clearly, the check extension as described in the previous section cannot be used to this end, since a user is identified as soon as he spents the check a second time, even if he honestly intends to use only the unspent part.

To achieve divisibility, we modify the check extension therefore in the following way: instead of having one identity part and several denominations, we build a check as a product of k denomination-terms, each of which has its own identity-part. For example, a divisible check looks like

$$m = (g_1^{u_1 s} g_2^{u_2 s} d_1) \cdot (e_1^{a_1} f_1^{b_1} h_1^{a_1 s}) \cdots (e_k^{a_k} f_k^{b_k} h_k^{a_k s}) d_2^{a_{k+1}}.$$

In the payment protocol, the user releases for the denominations that he wishes to spend, the points a_i, b_i, and lines $a_i s$. For the unspent parts, he releases "no information" (clearly, if we use method 2 for the payment protocol, it is not immediately clear how to do this).

When the user spends the blinded divisible check m^r at various places, using only the unspent parts, the bank (after deposit) can keep track of which parts have been spent already (A, B are always the same). As soon as a user double-spends a specific denomination, the blinded form $a_i sr$ of $a_i s$ is revealed (since this information was released in the form of a line). Because the blinded forms $a_i r$ of a_i have been released, as well as the blinded forms $u_1 sr, u_2 sr, r$ of $u_1 s, u_2 s$ and 1, the bank can compute s and hence the identity of the user. Observe that the exponents of the h_i's must depend on at least one of the other exponents a_i, b_i in the same denomination part: if this were not the case, the identity could be extracted as soon as two denomination-parts that are spent once are known to the bank. Nevertheless, other ways in which to encode the information needed to identify double-spenders are also possible.

Since there must be a certain relation between the exponents in each separate denomination term, in the withdrawal protocol for divisible money (which remains in essence the same as that for checks) the user (sending the denomination-part in the first transmission) has to prove that this knowledge is correctly formed. Note that the user can even request a refund for the unspent part of the divisible check.

There is one drawback of the above, in that all the payments made with the same check are linkable. We do not know how to dispose of this.

15 An analysis of efficiency

We here make some remarks on the efficiency of the cash system. A detailed analysis will appear in the full paper.

First of all, the user in the withdrawal protocol can precompute many numbers. Specifically, he can generate the random numbers s, u, v beforehand, and hence can precompute all of the following numbers (we here also assume that the user knows g_1^x, g_2^x, d_i^x beforehand):

$$m, m', z', g^v, (m')^v, su, A, B, x_1, x_2, y_1, y_2, z_1, z_2.$$

Hence only a', b' need (partly) be computed as well as c', c. Not counting the verification (since this is actually post-computation), a user therefore need only do approximately 2 exponentiations and computation of one hash-value in real time. The bank can precompute everything except for the response. Furthermore, if we use k-show signatures, essentially no extra real-time computing time is needed, due to the vector addition chain techniques. Therefore, all the multi-show extensions remain very efficient.

In the payment protocol, the user need perform only three multiplications (k in general) in real time, which is highly efficient. The computational effort of the shop, which must perform some verifications, is approximately four exponentiations.

In the extension to checks, the computation time for both the shop and the user increases only linearly. However, it seems that quite a lot of points and lines have to be released ($2(k+j)$ if j is the number of denominations that are spent in a check with k denomination terms). However, if on the average half of the denomination terms of a check are spent, a simple calculation (in which only the number of elements involved in the deposit is counted) learns that if there are at least five denominations terms in a check, then it is more efficient to use a check than separate coins. So our check extension actually is more efficient already for amounts that can only be paid with at least three diffferent coins (not considering the extra workload from the refund and withdrawal protocols).

An idea which may increase efficiency even further is that users do not need to compute all their random numbers in the withdrawal protocol (this especially applies to checks and divisibility) from $\mathbb{Z}_q$, but can restrict part of them to a smaller range. The exponentiations then require even less computational effort. This need not diminish privacy at all, if the information that must remain hidden is a function of several indices (this idea is inspired by the fact that the product – or sum – of k elements from $\mathbb{Z}_q^*$ is uniformly distributed over $\mathbb{Z}_q^*$ if at least one of the elements is uniformly chosen from $\mathbb{Z}_q^*$).

16 Off-line cash in wallets with observers

In this section we discuss how to transplant our cash system into a setting based on wallets with observers. Our system is the first that can be incorporated without great difficulty into this setting in such a way that even shared information is prevented. This is a due to the flexible structure of the representation problem, which allows all protocols used in our system (including the extensions) to be diverted. As we explained in Subsection 2.2, the important extra benefit one gets from this setting is that double-spending can be prevented (by the observer) rather than being detected after the fact. We would like to have a system that closely resembles our basic cash system, and it should preferably be such that if the observer is broken even before the account of its owner at the bank has been established, we still have the correctness of the basic cash system.

In order to achieve prevention of double-spending, the information that a user withdraws must be divided between the user-module and the observer in such a way that the user cannot spend it by himself. For our basic cash system, this amounts to making sure that the user by himself does not know a representation of m', yet the observer and the user together do know one. To this end, we first observe that if the observer $\mathcal{O}$ knows one representation of m_1 with respect to a certain generator-tuple, and the user(-module) $\mathcal{U}$ one of m_2, then only together they know a representation of $m_1 m_2$ (compare this to the protocol of Subsection 10.1).

16.1 Diverting the protocol for proving knowledge of a representation

Before we discuss a protocol which enables $\mathcal{O}$ and $\mathcal{U}$ together to prove knowledge of a representation of $m_1 m_2$ to a verifier $\mathcal{V}$, we add the extra requirements that $\mathcal{O}$ should not be able to leak any information about his particular representation of m_1 and need not even know m_2 in order to perform his part of the protocol. These requirements are imposed upon us (see [17]) if we want to avoid shared information.

The following protocol meets these requirements, and is the diverted version of the protocol described in Section 8 for proving knowledge of a representation. We assume that at the start of this protocol (see Figure 12), $\mathcal{O}$ knows the representation $(x_1, \ldots, x_k)$ of m_1 with respect to $(g_1, \ldots, g_k)$, whereas $\mathcal{U}$ knows the representation $(y_1, \ldots, y_k)$ of m_2 and also $m = m_1 m_2$.

Step 1 $\mathcal{O}$ generates at random k numbers $v_i \in_{\mathcal{R}} \mathbb{Z}_q$, computes $A' = \prod_{i=1}^{k} g_i^{v_i}$ and sends A' to $\mathcal{U}$.

Step 2 $\mathcal{U}$ generates at random k numbers $w_i \in_{\mathcal{R}} \mathbb{Z}_q$ and a number $d \in_{\mathcal{R}} \mathbb{Z}_q$. He then computes $A = A' \cdot (\prod_{i=1}^{k} g_i^{w_i}) \cdot m_1^d$ and sends A to $\mathcal{V}$.

Step 3 $\mathcal{V}$ generates uniformly at random a challenge $c \in_{\mathcal{R}} \mathbb{Z}_q$ and sends it to $\mathcal{V}$.

Step 4 $\mathcal{U}$ computes $c' = c + d \bmod q$ and sends challenge c' to $\mathcal{O}$.

Step 5 As in the basic proof of knowledge, $\mathcal{O}$ responds with the k numbers $r'_i = v_i + c'x_i \bmod q$.

Step 6 $\mathcal{U}$ verifies whether the responses of $\mathcal{O}$ are correct, i.e. whether $\prod_{i=1}^{k} g_i^{r'_i} = A' m_1^{c'}$. If this is the case, $\mathcal{U}$ computes, for $i \in \{1, \ldots, k\}$, the responses $r_i = r'_i + w_i + c y_i \bmod q$, and sends them to $\mathcal{V}$.

$\mathcal{V}$ accepts if and only if $\prod_{i=1}^{k} g_i^{r_i} = A m^c$.

It is easy to prove that the following proposition holds.

PROPOSITION 15 *$\mathcal{V}$ accepts if and only if $\mathcal{O}$ and $\mathcal{U}$ follow the protocol.*

We can consider this three-party protocol as being a special combination of two sub-protocols, one between $\mathcal{O}$ and $\mathcal{U}$, and the other between $\mathcal{U}$ and $\mathcal{V}$. Each of these two sub-protocols corresponds precisely to the two-party protocol for proving knowledge of a representation described in Section 8.

$$
\begin{array}{lclcr}
\mathcal{O} & & \mathcal{U} & & \mathcal{V} \\[1em]
(m_1 = \prod_{i=1}^{k} g_i^{x_i}) & & (m_2 = \prod_{i=1}^{k} g_i^{y_i}) & & \\
 & & & \xrightarrow{m(= m_1 m_2)} & \\
 & & \vdots & & \\[1em]
v_i \in_{\mathcal{R}} \mathbb{Z}_q & & & & \\
A' \leftarrow \prod_{i=1}^{k} g_i^{v_i} & \xrightarrow{\quad A' \quad} & & & \\
 & & w_i \in_{\mathcal{R}} \mathbb{Z}_q & & \\
 & & d \in_{\mathcal{R}} \mathbb{Z}_q & & \\
 & & A \leftarrow A' \cdot (\prod_{i=1}^{k} g_i^{w_i}) \cdot m_1^d & \xrightarrow{\quad A' \quad} & \\[1em]
 & & & \xleftarrow{\quad c \quad} & c \in_{\mathcal{R}} \mathbb{Z}_q \\
 & \xleftarrow{\quad c' \quad} & c' \leftarrow c + d \bmod q & & \\[1em]
c' \stackrel{?}{\neq} 0 & & & & \\
r_i' \leftarrow v_i + c' x_i \bmod q & \xrightarrow{r_1, \ldots, r_k} & & & \\
 & & \prod_{i=1}^{k} g_i^{r_i'} \stackrel{?}{=} A' m_1^{c'} & & \\
 & & r_i \leftarrow r_i' + w_i + c y_i \bmod q & \xrightarrow{r_1, \ldots, r_k} & \\
 & & & & \prod_{i=1}^{k} g_i^{r_i} \stackrel{?}{=} A m^c
\end{array}
$$

FIGURE 12. Diverting the protocol for proving knowledge of a representation

PROPOSITION 16 *If $\mathcal{U}$ follows the protocol, then transcripts of the sub-protocol between $\mathcal{O}$ and $\mathcal{U}$ are statistically independent of transcripts of the sub-protocol between $\mathcal{U}$ and $\mathcal{V}$.*

That is, no shared information can arise in this protocol (see [17]). The simple proof of this will be given in the full paper.

16.2 A first attempt to transplant the basic two-party cash system

From this protocol to diverting the basic payment protocol seems only a minor step. However, there are some difficulties that one encounters when trying to meet all the requirements of the wallet-with-observer setting. We therefore first discuss an attempt which unfortunately leads us to a dead-end street. This will give the reader a feeling of how to proceed in order to avoid the main difficulties.

In the basic cash scheme, the user withdraws a number m' with the signature of the bank, such that he has committed to a splitting $m' = AB$. In the three-party setting, each of A, B must be split up between $\mathcal{O}$ and $\mathcal{U}$. That is, $A = A_{\mathcal{O}}A_{\mathcal{U}}$ and $B = B_{\mathcal{O}}B_{\mathcal{U}}$, with one representation of $A_{\mathcal{O}}$ and one of $B_{\mathcal{O}}$ known only to $\mathcal{O}$ and one representation of $A_{\mathcal{U}}$ and one of $B_{\mathcal{U}}$ known only to $\mathcal{U}$. We have to choose this specific division of knowledge since, in order to prevent shared information, $\mathcal{O}$ may not even know either of A, B.

To achieve the statistical independence property in the diverted payment protocol, the precise form of A and B must unavoidably be somewhat more complicated. Specifically, if we let

$$\begin{aligned} A_{\mathcal{O}} &= g_1^{o_1} g_2^{o_2} g_3^{o_3}, \\ B_{\mathcal{O}} &= g_1^{o_4} g_2^{o_5} g_3^{o_6}, \\ A_{\mathcal{U}} &= g_1^{u_1} g_2^{u_2} g_3^{u_3}, \\ B_{\mathcal{U}} &= g_1^{u_4} g_2^{u_5} g_3^{u_6}, \end{aligned}$$

we then take

$$\begin{aligned} A &= A_{\mathcal{O}} A_{\mathcal{U}} B_{\mathcal{O}}^d \\ B &= B_{\mathcal{O}} B_{\mathcal{U}}. \end{aligned}$$

The number $d \in \mathbb{Z}_q$ is a random choice of the user that must remain unknown to $\mathcal{O}$. With this split-up, it turns out that the payment protocol can be diverted. To this end, in the withdrawal protocol AB must be withdrawn in such a way that the above division of knowledge of a representation of AB is accomplished. That is, the observer must end up knowing only a representation (o_1, o_2, o_3) of $A_{\mathcal{O}}$ and a representation (o_4, o_5, o_6) of $B_{\mathcal{O}}$ with respect to (g_1, g_2, g_3), and nothing more. The user-module must end up knowing a representation (u_1, u_2, u_3) of $A_{\mathcal{U}}$, a representation (u_4, u_5, u_6) of $B_{\mathcal{U}}$ with respect to (g_1, g_2, g_3), the numbers $A_{\mathcal{O}}$, $B_{\mathcal{O}}$, and the signature $\text{sign}(A, B)$ of the bank on $A = A_{\mathcal{O}} A_{\mathcal{U}} B_{\mathcal{O}}^d$, $B = B_{\mathcal{O}} B_{\mathcal{U}}$.

It is clear from this split-up that the user-module on its own does not know a representation of either of A, B, and hence cannot spend this information without cooperation of the observer. The identity of the user must be encoded in the representations in a suitable way, if the requirement that the identity of a double-spender (who hence broke the tamper-resistance) is to be revealed. We return to how to do this after showing the diverted payment protocol, since it depends on the form of the verification relation of that protocol how the identity must be encoded (however, it will turn out that there is no nice way to encode the identity, i.e. we are already in the dead-end street).

$$
\begin{array}{lclcr}
\mathcal{O} & & \mathcal{U} & & \mathcal{S} \\
(A_{\mathcal{O}} = g_1^{o_1} g_2^{o_2} g_3^{o_3}, & & (A_{\mathcal{U}} = g_1^{u_1} g_2^{u_2} g_3^{u_3}, & & \\
B_{\mathcal{O}} = g_1^{o_6} g_2^{o_5} g_3^{o_6}) & & B_{\mathcal{U}} = g_1^{u_6} g_2^{u_5} g_3^{u_6}) & & \\
 & & & \xrightarrow{A, B, \mathrm{sign}(A,B)} & \\
 & & & \xleftarrow{c} & c \in \mathbb{Z}_q^* \\
 & & d \in_{\mathcal{R}} \mathbb{Z}_q^* & & \\
 & \xleftarrow{c'} & c' \leftarrow c + d \bmod q & & \\
c' \stackrel{?}{\neq} 0 & & & & \\
r'_1 \leftarrow o_1 + c' o_4 \bmod q & & & & \\
r'_2 \leftarrow o_2 + c' o_5 \bmod q & & & & \\
r'_3 \leftarrow o_3 + c' o_6 \bmod q & \xrightarrow{r'_1, r'_2, r'_3} & & & \\
 & & g_1^{r'_1} g_2^{r'_2} g_3^{r'_3} \stackrel{?}{=} A_{\mathcal{O}} B_{\mathcal{O}}^{c'} & & \\
 & & r_1 \leftarrow r'_1 + u_1 + c u_4 \bmod q & & \\
 & & r_2 \leftarrow r'_2 + u_2 + c u_5 \bmod q & & \\
 & & r_3 \leftarrow r'_3 + u_3 + c u_6 \bmod q & \xrightarrow{r_1, r_2, r_3} & \\
 & & & & g_1^{r_1} g_2^{r_2} g_3^{r_3} \stackrel{?}{=} AB^c
\end{array}
$$

FIGURE 13. A first attempt to achieve a suitable diverted payment protocol

The diverted payment protocol, from which it will be clear why A must be of this somewhat more complicated form, is as follows (see Figure 13):

Step 1 $\mathcal{U}$ sends the triple A, B, $\mathrm{sign}(A, B)$ to the shop $\mathcal{S}$. (In case there are several denomiations, $\mathcal{U}$ also has to inform $\mathcal{S}$ of the amount of money the coin is supposedly worth.)

Step 2 $\mathcal{S}$ verifies the signature as in the basic cash system. If the verification holds, $\mathcal{S}$ sends a challenge $c \in \mathbb{Z}_q$ to $\mathcal{U}$.

Step 3 $\mathcal{U}$ computes $c' = c + d \bmod q$, and sends the challenge c' to $\mathcal{O}$.

Step 4 $\mathcal{O}$ computes the responses $r'_1 = o_1 + c' o_4 \bmod q$, $r'_2 = o_2 + c' o_5 \bmod q$ and $r'_3 = o_3 + c' o_6 \bmod q$, and sends these to $\mathcal{U}$.

Step 5 $\mathcal{U}$ verifies whether $g_1^{r'_1} g_2^{r'_2} g_3^{r'_3} \stackrel{?}{=} A_{\mathcal{O}} B_{\mathcal{O}}^{c'}$. If this verification holds, he computes $r_1 = r'_1 + u_1 + c u_4 \bmod q$, $r_2 = r'_2 + u_2 + c u_5 \bmod q$ and $r_3 = r'_3 + u_3 + c u_6 \bmod q$, and sends these responses to $\mathcal{S}$.

$\mathcal{S}$ accepts if and only if $g_1^{r_1} g_2^{r_2} g_3^{r_3} = AB^c$.

Again, as in the basic cash protocol, we can increase efficiency even a little further if it is sufficient that framing is merely infeasible, by using only two generators instead of three. The same remarks as in the basic cash system concerning the choice of the challenge c of the shop hold.

As with the diverted protocol for proving knowledge of a representation, one can easily prove the following two propositions for this payment protocol.

PROPOSITION 17 *$\mathcal{S}$ accepts if and only if $\mathcal{O}$ and $\mathcal{U}$ follow the protocol.*

PROPOSITION 18 *If $\mathcal{U}$ follows the protocol, then transcripts of the sub-protocol between $\mathcal{O}$ and $\mathcal{U}$ are statistically independent of transcripts of the sub-protocol between $\mathcal{U}$ and $\mathcal{S}$.*

This ensures that the privacy of the user is unconditionally guaranteed since, as in the basic cash system, the numbers AB and the signature are also statistically independent from the information that the bank sees. If the user wants to spend the same money a second time, the observer simply refuses to cooperate in the payment protocol. So in order to double-spend, the user has to break the tamper-resistance of the observer and extract $o_1, \ldots, o_6$.

Nevertheless, even in that case he must be identified after the fact, as in the basic cash system. This is where we run into troubles. To see this, observe that the identity of the user must have been encoded in such a way that two sets of responses (r_1, r_2, r_3), (r_1', r_2', r_3') to two different challenges c respectively c' reveal the identity of the user. We first investigate what equations the bank can extract from double-spending. We have by the verification relations

$$\begin{aligned} g_1^{r_1} g_2^{r_2} g_3^{r_3} &= AB^c, \\ g_1^{r_1'} g_2^{r_2'} g_3^{r_3'} &= AB^{c'}. \end{aligned}$$

Hence, it follows that $B = g_1^{(r_1 - r_1')/(c-c')} g_2^{(r_2 - r_2')/(c-c')} g_3^{(r_3 - r_3')/(c-c')}$. Since B is of the form $B = B_{\mathcal{O}} B_{\mathcal{U}} = g_1^{o_4+u_4} g_2^{o_5+u_5} g_3^{o_6+u_6}$, it must be that

$$\begin{aligned} o_4 + u_4 &= (r_1 - r_1')/(c - c') \bmod q, \\ o_5 + u_5 &= (r_2 - r_2')/(c - c') \bmod q, \\ o_6 + u_6 &= (r_3 - r_3')/(c - c') \bmod q. \end{aligned}$$

In the same way, also

$$\begin{aligned} o_1 + u_1 + do_4 &= (c' r_1 - c r_1')/(c' - c) \bmod q, \\ o_2 + u_2 + do_5 &= (c' r_2 - c r_2')/(c' - c) \bmod q, \\ o_1 + u_3 + do_6 &= (c' r_3 - c r_3')/(c' - c) \bmod q \end{aligned}$$

can be computed from two different executions of the payment protocol.

Therefore, we must make sure that in the withdrawal protocol the identifying information of the user is encoded in the representations in such a way that from these six equations the bank can compute the identifying information of the double-spender. The splitting that must be made in the withdrawal protocol, dictated by this diverted payment scheme, is

$$AB = \overbrace{A_{\mathcal{O}} A_{\mathcal{U}} B_{\mathcal{O}}^d}^{A} \cdot \overbrace{B_{\mathcal{O}} B_{\mathcal{U}}}^{B}.$$

We make the simple observation that if the user and the observer together (in a two-party protocol) come up with AB as above, and the user wants to have this signed thereafter by the bank (in a two-party protocol) in order to end up with information that can be used to pay, he had better not blind AB. If he does so, the payment protocol will not function properly sinds the responses that the observer gives only function for AB. As a consequence, if the user makes a random choice $s \in \mathbb{Z}_q$, and sends ABg^s to the bank in order to have it signed, he had better unblind this number to AB (he can do so by Proposition 12) if he wants to be able to pay with this information at a shop.

In view of the above, it seems that we must construct the withdrawal protocol as follows. First, the observer and the user-module together determine $A_{\mathcal{O}}$, $A_{\mathcal{U}}$, $B_{\mathcal{O}}$ and $B_{\mathcal{U}}$ in such a way that the identity of the user is encoded in a suitable way, and A, B must remain unknown to the observer. Hereto, $B_{\mathcal{U}}$ must be chosen at random by the user-module in such a way that it remains unknown to the observer. To ensure that A remains unknown to the observer, either d or $\mathcal{A}_{\mathcal{U}}$ must be randomly chosen by the user-module. Then, the user-module generates at random two numbers $s, d \in \mathbb{Z}_q$, and sends $m_1 = A_{\mathcal{U}} B_{\mathcal{U}} B_{\mathcal{O}}^d g^s$ to the observer. Clearly, there must be some proof of correctness of m_1 here. If the observer accepts this proof, it computes $m = m_1 A_{\mathcal{O}} B_{\mathcal{O}}$ and signs m with its native signature scheme (see [17]). This finishes the protocol with the observer. Next, the user-module sends m together with the signature of the observer to the bank. The bank, after having verified the correctness of this signature, then performs the basic withdrawal protocol with the user-module, as described in Section 11. Hereby, the user-module transforms m to $m' = mg^{-s}$.

It is easy to see from that if the user-module follows these actions, it ends up with a signature of the bank on (A, B), such that $A = A_{\mathcal{O}} A_{\mathcal{U}} B_{\mathcal{O}}^d$ and $B = B_{\mathcal{O}} B_{\mathcal{U}}$. Furthermore, both A, B are not only unknown but also statistically independent from the view of the observer because of the two random choices $d, s \in \mathbb{Z}_q$ made by the user-module. However, and this is what goes wrong, a detailed study will reveal that there is no way (without this amounting to a lot of patching, that is) to encode the user's identity in $B_{\mathcal{O}}$, $B_{\mathcal{U}}$, basically because $B_{\mathcal{U}}$ must be blinded by the user.

16.3 How to avoid these problems

The attempt of the previous subsection failed. This is actually due to the fact that there is but little flexibility left to encode the identity in a way that meets all requirements. So, in order to get greater flexibility, we start looking again at the diverted payment protocol. It seems that this is where the difficulties started, because all further relations were imposed by the form of the payment protocol. Therefore, we first show how to divert the payment protocol in a very general way. To this end, let $\alpha, \beta, \gamma, \delta, \epsilon$ be elements from $\mathbb{Z}_q$. As before, let

$$A_{\mathcal{O}} = g_1^{o_1} g_2^{o_2} g_3^{o_3}, \quad B_{\mathcal{O}} = g_1^{o_4} g_2^{o_5} g_3^{o_6}$$
$$A_{\mathcal{U}} = g_1^{u_1} g_2^{u_2} g_3^{u_3}, \quad B_{\mathcal{U}} = g_1^{u_4} g_2^{u_5} g_3^{u_6}.$$

We now take the general form

$$A = A_{\mathcal{O}}^{\alpha} A_{\mathcal{U}}^{\beta} B_{\mathcal{O}}^{\gamma}$$
$$B = B_{\mathcal{O}}^{\delta} B_{\mathcal{U}}^{\epsilon}.$$

A suitable choice of values of the parameters will be made *after* we decide how to encode the identity. That is, we will work backwards and choose values of these five parameters in the end.

$$
\begin{array}{lcl}
\mathcal{O} & \mathcal{U} & \mathcal{S} \\
(A_{\mathcal{O}} = g_1^{o_1} g_2^{o_2} g_3^{o_3}, & (A_{\mathcal{U}} = g_1^{u_1} g_2^{u_2} g_3^{u_3}, & \\
B_{\mathcal{O}} = g_1^{o_4} g_2^{o_5} g_3^{o_6}) & B_{\mathcal{U}} = g_1^{u_4} g_2^{u_5} g_3^{u_6}) & \\
 & \xrightarrow{A, B, \mathrm{sign}(A,B)} & \\
 & \xleftarrow{c} & c \in \mathbb{Z}_q \\
\xleftarrow{c'} & c' \leftarrow (\delta c + \gamma)/\alpha \bmod q & \\
r'_1 \leftarrow o_1 + c' o_4 \bmod q & & \\
r'_2 \leftarrow o_2 + c' o_5 \bmod q & & \\
r'_3 \leftarrow o_3 + c' o_6 \bmod q \xrightarrow{r'_1, r'_2, r'_3} & & \\
 & g_1^{r'_1} g_2^{r'_2} g_3^{r'_3} \stackrel{?}{=} A_{\mathcal{O}} B_{\mathcal{O}}^{c'} & \\
 & r_1 \leftarrow \alpha r'_1 + \beta u_1 + \epsilon c u_4 \bmod q & \\
 & r_2 \leftarrow \alpha r'_2 + \beta u_2 + \epsilon c u_5 \bmod q & \\
 & r_3 \leftarrow \alpha r'_3 + \beta u_3 + \epsilon c u_6 \bmod q \xrightarrow{r_1, r_2, r_3} & \\
 & & g_1^{r_1} g_2^{r_2} g_3^{r_3} \stackrel{?}{=} A B^c
\end{array}
$$

FIGURE 14. Diverting a generalized form of the payment protocol

The diverted payment scheme for this general choice of A, B, is as follows (see Figure 14):

Step 1 $\mathcal{U}$ sends the triple A, B, $\mathrm{sign}(A, B)$ to the shop $\mathcal{S}$.

Step 2 $\mathcal{S}$ verifies the signature as in the basic cash system. If the verification holds, $\mathcal{S}$ sends a challenge $c \in \mathbb{Z}_q$ to $\mathcal{U}$.

Step 3 $\mathcal{U}$ computes $c' = (\delta c + \gamma)/\alpha \bmod q$, and sends the challenge c' to $\mathcal{O}$.

Step 4 $\mathcal{O}$ computes the responses $r'_1 = o_1 + c' o_4 \bmod q$, $r'_2 = o_2 + c' o_5 \bmod q$ and $r'_3 = o_3 + c' o_6 \bmod q$, and sends these to $\mathcal{U}$.

Step 5 $\mathcal{U}$ verifies whether $g_1^{r'_1} g_2^{r'_2} g_3^{r'_3} \stackrel{?}{=} A_{\mathcal{O}} B_{\mathcal{O}}^{c'}$. If this verification holds, he computes $r_1 = \alpha r'_1 + \beta u_1 + \epsilon c u_4 \bmod q$, $r_2 = \alpha r'_2 + \beta u_2 + \epsilon c u_5 \bmod q$ and $r_3 = \alpha r'_3 + \beta u_3 + \epsilon c u_6 \bmod q$, and sends these responses to $\mathcal{S}$.

$\mathcal{S}$ accepts if and only if $g_1^{r_1} g_2^{r_2} g_3^{r_3} = AB^c$.

Again, we can easily prove the next two propositions.

PROPOSITION 19 *$\mathcal{S}$ accepts if and only if $\mathcal{O}$ and $\mathcal{U}$ follow the protocol.*

PROPOSITION 20 *If $\mathcal{U}$ follows the protocol, then transcripts of the sub-protocol between $\mathcal{O}$ and $\mathcal{U}$ are statistically independent of transcripts of the sub-protocol between $\mathcal{U}$ and $\mathcal{S}$.*

Observe that we used three parameters to blind c to c', yet only one is actually needed for the latter proposition to hold. It will turn out that the specific blinding of c is quite different than that used in the previous subsection.

As before, the bank must be able to identify a double-spender (who hence broke the tamper-resistance of the observer), and so the information that the bank can extract from two different sets of responses (r_1, r_2, r_3), (r'_1, r'_2, r'_3), corresponding to two different challenges c respectively c', must reveal the identity of the user. Again, from the verification relations

$$\begin{aligned} g_1^{r_1} g_2^{r_2} g_3^{r_3} &= AB^c, \\ g_1^{r'_1} g_2^{r'_2} g_3^{r'_3} &= AB^{c'}, \end{aligned}$$

the bank can extract

$$\begin{aligned} \delta o_4 + \epsilon u_4 &= (r_1 - r'_1)/(c - c') \bmod q, \\ \delta o_5 + \epsilon u_5 &= (r_2 - r'_2)/(c - c') \bmod q, \\ \delta o_6 + \epsilon u_6 &= (r_3 - r'_3)/(c - c') \bmod q \end{aligned}$$

and

$$\begin{aligned} \alpha o_1 + \beta u_1 + \gamma o_4 &= (c' r_1 - c r'_1)/(c' - c) \bmod q, \\ \alpha o_2 + \beta u_2 + \gamma o_5 &= (c' r_2 - c r'_2)/(c' - c) \bmod q, \\ \alpha o_1 + \beta u_3 + \gamma o_6 &= (c' r_3 - c r'_3)/(c' - c) \bmod q. \end{aligned}$$

This must reveal the identifying information of the double-spender. We now will make the following choice for the five parameters:

$$\alpha = \beta, \quad \gamma = -\delta, \quad \epsilon = 0.$$

This ensures that almost all parameters can be cancelled out by adding certain equations. Namely,

$$\begin{aligned} (\delta o_4 + \epsilon u_4) + (\alpha o_1 + \beta u_1 + \gamma o_4) &= \alpha(o_1 + u_1) \bmod q, \\ (\delta o_5 + \epsilon u_5) + (\alpha o_2 + \beta u_2 + \gamma o_5) &= \alpha(o_1 + u_1) \bmod q, \\ (\delta o_6 + \epsilon u_6) + (\alpha o_3 + \beta u_3 + \gamma o_6) &= \alpha(o_1 + u_1) \bmod q \end{aligned}$$

for these specific values of the parameters. Therefore, the bank can compute from the two sets of responses the following three numbers:

$$\begin{aligned} (r_1 - r'_1)/(c - c') + (c' r_1 - c r'_1)/(c' - c) &= \alpha(o_1 + u_1) \bmod q, \\ (r_2 - r'_2)/(c - c') + (c' r_2 - c r'_2)/(c' - c) &= \alpha(o_2 + u_2) \bmod q, \\ (r_3 - r'_3)/(c - c') + (c' r_3 - c r'_3)/(c' - c) &= \alpha(o_3 + u_3) \bmod q. \end{aligned}$$

Now we compare this with the basic (two party) cash system, and let the α here replace the s used there to blind $m = AB$ to $m' = m^s$, and replace the $(u_1, u_2, 1)$ of the basic cash scheme by $(o_1 + u_1, o_2 + u_2, (o_3 + u_3 =)1)$. Note that the fact that $u_1 + o_1$ must equal 1, implies that there is no sense in sharing this between $\mathcal{O}$ and $\mathcal{U}$.

It turns out that this is precisely a choice of parameters with which we can meet all the requirements of the observer setting in such a way that all the protocols closely resemble those of the basic two-party setting. What we will do is have $\mathcal{O}$ and $\mathcal{U}$ generate uniformly at random a number $I = g_1^{o_1+u_1} g_2^{o_2+u_2}$. We let

$$\begin{aligned} A_{\mathcal{O}} &= g_1^{o_1} g_2^{o_2} \\ B_{\mathcal{O}} &= g_1^{o_3} g_2^{o_4} \\ A_{\mathcal{U}} &= g_1^{u_1} g_2^{u_2} \end{aligned}$$

and

$$A = (A_{\mathcal{O}} A_{\mathcal{U}})^{\alpha} B_{\mathcal{O}}^{\beta}$$
$$B = B_{\mathcal{O}}^{-\beta}.$$

Due to our choice of the parameters, and this is very important, $B_{\mathcal{O}}$ drops out when forming the product AB (that is, $AB = (A_{\mathcal{O}} A_{\mathcal{U}})^{\alpha}$). Moreover, we no longer need $B_{\mathcal{U}}$ (although $\mathcal{U}$ can multiply it into the splitting, it does not bring him anything). We deal with g_3 separately, since its exponent need not be divided between $\mathcal{O}$ and $\mathcal{U}$.

We now show the correct form of our cash system in wallets with observers, using the previous ideas.

16.4 The set-up phase of the system

When the user wants to open an account at the bank $\mathcal{B}$ the following protocol is executed (see Figure 15):

Step 1 $\mathcal{U}$ generates at random three numbers $u_1, u_2, t \in_{\mathcal{R}} \mathbb{Z}_q$. $\mathcal{U}$ computes $A_{\mathcal{U}} = g_1^{u_1} g_2^{u_2}$, and sends $T' = A_{\mathcal{U}} g^t$ to $\mathcal{O}$. Note that t's function is merely to blind $A_{\mathcal{U}}$.

Step 2 $\mathcal{O}$ generates at random two numbers $o_1, o_2 \in_{\mathcal{R}} \mathbb{Z}_q$ and computes $A_{\mathcal{O}} = g_1^{o_1} g_2^{o_2}$. He then signs $T = T' A_{\mathcal{O}}$ with his native signature scheme (see [17]), and sends both $A_{\mathcal{O}}$ and the signature to $\mathcal{U}$.

Step 3 $\mathcal{U}$ verifies the correctness of the signature and, if this verification holds, sends $T = T' A_{\mathcal{O}}$, the signature of $\mathcal{O}$, and t to $\mathcal{B}$.

Step 4 $\mathcal{B}$ verifies the signature. If the verification holds, $\mathcal{B}$ computes T/g^t. Observe that if $\mathcal{U}$ was honest, this equals $g_1^{o_1+u_1} g_2^{o_2+u_2}$.

Step 5 $\mathcal{O}$ and $\mathcal{U}$ together proof knowledge of a representation of T/g^t with respect to (g_1, g_2). To this end, they use the protocol of Subsection 16.1 (with $k = 2$).

If $\mathcal{B}$ accepts the proof of knowledge of $\mathcal{O}$ and $\mathcal{U}$, it stores $I = T/g^t$ together with the user's real identity with his new account.

Note that the signature of $\mathcal{O}$ guarantees to $\mathcal{B}$ that $\mathcal{U}$ did not simulate the part of $\mathcal{O}$ in the protocol. Furthermore, Step 5 can actually be left out since it also has to be performed when the user wants to withdraw a coin in a later stage. That is, if $\mathcal{U}$ does not know a representation of his part $A_{\mathcal{U}}$ of I, he will never be able to succesfully enter the withdrawal protocol.

Informally, what has happened in this protocol is that $\mathcal{O}$ has ended up knowing $A_{\mathcal{O}} = g_1^{o_1} g_2^{o_2}$ and the secret information (o_1, o_2), and $\mathcal{U}$ knowing $A_{\mathcal{U}} = g_1^{u_1} g^{u_2}$. Moreover, $\mathcal{O}$ has no information about $A_{\mathcal{U}}$ whatsoever, whereas $\mathcal{U}$ also knows $A_{\mathcal{O}}$ (but not o_1, o_2). The bank has listed I as the identifying information of the user, and due to $\text{sign}_{\mathcal{O}}(T)$ can be assured that $\mathcal{U}$ by itself does not know a representation of I. It is easy to prove that that if and only if $\mathcal{U}$ follows the protocol, the bank accepts, and the above split-up of the representation of $A_{\mathcal{O}} A_{\mathcal{U}}$ is established.

In case the user breaks open the observer and double-spends the same cash, the bank will be able to find out $(o_1 + u_1, o_2 + u_2)$. By computing $I = g_1^{o_1+u_1} g_2^{o_2+u_2}$, he can then determine the double-spender's identity.

$$
\begin{array}{ccccc}
\mathcal{O} & & \mathcal{U} & & \mathcal{B} \\
 & & u_1, u_2 \in_{\mathcal{R}} \mathbb{Z}_q & & \\
 & & t \in_{\mathcal{R}} \mathbb{Z}_q & & \\
 & & A_{\mathcal{U}} \leftarrow g_1^{u_1} g_2^{u_2} & & \\
 & \xleftarrow{\quad T' \quad} & T' \leftarrow A_{\mathcal{U}} g^t & & \\
o_1, o_2 \in_{\mathcal{R}} \mathbb{Z}_q & & & & \\
A_{\mathcal{O}} \leftarrow g_1^{o_1} g_2^{o_2} & & & & \\
T \leftarrow T' A_{\mathcal{O}} & \xrightarrow{A_{\mathcal{O}}, \mathrm{sign}_{\mathcal{O}}(T)} & & & \\
 & & T \leftarrow T' A_{\mathcal{O}} & & \\
 & & \text{Verify } \mathrm{sign}_{\mathcal{O}}(T) & \xrightarrow{T, \mathrm{sign}_{\mathcal{O}}(T), t} & \\
 & & & & \text{Verify } \mathrm{sign}_{\mathcal{O}}(T) \\
 & & \text{(Diverted proof of knowledge of a representation of } T/g^t\text{)} & & \\
 & & \longleftrightarrow & &
\end{array}
$$

FIGURE 15. The set-up protocol for establishing an account

16.5 The withdrawal protocol

When a user wants to withdraw a coin from his account, $\mathcal{O}$ and $\mathcal{U}$ together first prove to $\mathcal{B}$ knowledge of a representation of I, using the protocol described in Subsection 16.1 with $k = 2$. If $\mathcal{B}$ accepts the proof of the wallet, then (for each coin that the user wishes to withdraw) the following withdrawal protocol is executed (see Figure 16):

Step 1 $\mathcal{O}$ generates at random two number $o_3, o_4 \in \mathbb{Z}_q$, and computes $B_{\mathcal{O}} = g_1^{o_3} g_2^{o_4}$. He then sends $B_{\mathcal{O}}$ to $\mathcal{U}$. We remark that, although this step is part of the protocol, $\mathcal{O}$ can actually send $B_{\mathcal{O}}$ at any time before $\mathcal{U}$ makes the splitting. $B_{\mathcal{O}}$ is a number that $\mathcal{O}$ uses in the payment protocol when $\mathcal{U}$ wants to pay with the information he withdraws.

Step 2 $\mathcal{U}$ and $\mathcal{B}$ perform the withdrawal protocol as in the basic cash system, with only one minor change. Namely, the splitting that $\mathcal{U}$ applies to the blinded number $m' = m^\alpha$ (note that we used the symbol s instead of α in the basic cash system – we use α here because it is more in line with the discussion so far) is not a completely random one (although the resulting splitting of m' has the same distribution as a random splitting). More specifically, with $m' = AB$, $\mathcal{U}$ computes A, B as $A = (A_{\mathcal{O}} A_{\mathcal{U}})^\alpha B_{\mathcal{O}}^\beta g_3^\gamma$, $B = B_{\mathcal{O}}^{-\beta} g_3^{\alpha-\gamma}$, with α, β, γ being random choices of his own, not equal to zero. We remark that $\mathcal{U}$ must store α, β, γ, since otherwise he will not be able to compute correct responses in the payment protocol. Furthermore, $\mathcal{U}$ can also multiply A by $g_1^{u_3} g_2^{u_4}$ (for randomly chosen u_3, u_4), and divide B by this term. However, this does not bring him anything at all, and since it only costs extra computations we leave this out.

$$
\begin{array}{lcccccr}
\mathcal{O} & & & \mathcal{U} & & & \mathcal{B} \\
(A_{\mathcal{O}} = g_1^{o_1} g_2^{o_2}) & & & (A_{\mathcal{U}} = g_1^{u_1} g_2^{u_2}) & & & (g, h = g^x) \\
 & & \multicolumn{3}{c}{\text{(Diverted proof of knowledge of a representation of } I\text{)}} & & \\
 & & & \longleftrightarrow & & & \\
 & & & \vdots & & & \\
o_3, o_4 \in_{\mathcal{R}} \mathbb{Z}_q & & & & & & \\
B_{\mathcal{O}} \leftarrow g_1^{o_3} g_2^{o_4} & \xrightarrow{B_{\mathcal{O}}} & & & & & \\
 & & & & & & w \in_{\mathcal{R}} \mathbb{Z}_q \\
 & & & & & & m \leftarrow I g_3 \\
 & & & & & & z \leftarrow m^x \\
 & & & & & & a \leftarrow g^w \\
 & & & & \xleftarrow{z, a, b} & & b \leftarrow m^w \\
 & & & m \leftarrow (A_{\mathcal{O}} A_{\mathcal{U}}) g_3 & & & \\
 & & & \alpha, \beta, \gamma \in_{\mathcal{R}} Z_q^* & & & \\
 & & & m' \leftarrow m^\alpha & & & \\
 & & & z' \leftarrow z^\alpha & & & \\
 & & & \text{Split } m' \text{ in } A, B\text{:} & & & \\
 & & & A \leftarrow (A_{\mathcal{O}} A_{\mathcal{U}})^\alpha B_{\mathcal{O}}^\beta g_3^\gamma & & & \\
 & & & B \leftarrow B_{\mathcal{O}}^{-\beta} g_3^{\alpha - \gamma} & & & \\
 & & & u, v \in_{\mathcal{R}} \mathbb{Z}_q & & & \\
 & & & a' \leftarrow a^u g^v & & & \\
 & & & b' \leftarrow b^{su} (m')^v & & & \\
 & & & c' \leftarrow \mathcal{H}(m', z', a', b', A) & & & \\
 & & & c \leftarrow c'/u \bmod q & \xrightarrow{c} & & \\
 & & & & \xleftarrow{r} & & r \leftarrow xc + w \bmod q \\
 & & & g^r \stackrel{?}{=} h^c a & & & \\
 & & & m^r \stackrel{?}{=} z^c b & & & \\
 & & & r' \leftarrow ru + v \bmod q & & &
\end{array}
$$

FIGURE 16. Withdrawal protocol for coins in wallets with observers

16.6 The payment protocol

From the withdrawal protocol, the user has ended up with numbers A, B, $\mathrm{sign}(A,B)$ and random numbers $\alpha, \beta \in \mathbb{Z}_q$, such that

$$A = (A_{\mathcal{O}} A_{\mathcal{U}})^{\alpha} B_{\mathcal{O}}^{\beta} g_3^{\gamma},$$
$$B = B_{\mathcal{O}}^{-\beta} g_3^{\alpha-\gamma}.$$

When the user wants to pay with the withdrawn information, the following protocol with shop $\mathcal{S}$ is followed (see Figure 17):

Step 1 $\mathcal{U}$ sends the triple A, B, $\mathrm{sign}(A,B)$ to $\mathcal{S}$.

Step 2 $\mathcal{S}$ verifies the signature as in the basic cash system. If the verification holds, $\mathcal{S}$ sends a challenge $c \in \mathbb{Z}_q^*$ to $\mathcal{U}$.

Step 3 $\mathcal{U}$ verifies that $c \neq 1$. If this is the case, $\mathcal{U}$ computes $c' = \beta(1-c)/\alpha \bmod q$, and sends the challenge c' to $\mathcal{O}$.

Step 4 First, $\mathcal{O}$ verifies that the coin the user wants to spend has not already been spent before. If this is the case, and also $c' \neq 0$, $\mathcal{O}$ computes the two responses $r_1' = o_1 + c'o_3 \bmod q$ and $r_2' = o_2 + c'o_4 \bmod q$ and sends these to $\mathcal{U}$.

Step 5 $\mathcal{U}$ verifies whether $g_1^{r_1'} g_2^{r_2'} \stackrel{?}{=} A_{\mathcal{O}} B_{\mathcal{O}}^{c'}$. If this verification holds he computes, using the responses of the observer, $r_1 = \alpha(r_1' + u_1) \bmod q$ and $r_2 = \alpha(r_2' + u_2) \bmod q$. The third response r_3 he can compute all by himself as $r_3 = \gamma + c(\alpha - \gamma) \bmod q$. He then sends these three responses to $\mathcal{S}$.

$\mathcal{S}$ accepts if and only if $g_1^{r_1} g_2^{r_2} g_3^{r_3} = AB^c$.

16.7 The deposit protocol

This is exactly the same as in the two-party system, and hence we refer the reader to Subsection 11.4. Now suppose that the bank detects that information A, B, $\mathrm{sign}(A,B)$ has been spent more than once. The bank then has at its disposal two sets of responses (r_1, r_2, r_3) and (r_1', r_2', r_3'), corresponding to two different challenges c and c'. By the verification relations, the following equations must hold:

$$g_1^{r_1} g_2^{r_2} g_3^{r_3} = AB^c,$$
$$g_1^{r_1'} g_2^{r_2'} g_3^{r_3'} = AB^{c'}.$$

As explained before, from these relations we get:

$$A = g_1^{(c'r_1 - cr_1')/(c'-c)} g_2^{(c'r_2 - cr_2')/(c'-c)} g_3^{(c'r_3 - cr_3')/(c'-c)},$$
$$B = g_1^{(r_1 - r_1')/(c-c')} g_2^{(r_2 - r_2')/(c-c')} g_3^{(r_3 - r_3')/(c'-c)}.$$

Now, we know that

$$A = (A_{\mathcal{O}} A_{\mathcal{U}})^{\alpha} B_{\mathcal{O}}^{\beta} g_3^{\gamma} = g_1^{\alpha(o_1+u_1)+\beta o_3} g_2^{\alpha(o_2+u_2)+\beta o_4} g_3^{\gamma},$$
$$B = B_{\mathcal{O}}^{-\beta} = g_1^{-\beta o_3} g_2^{-\beta o_4} g_3^{\alpha-\gamma}.$$

$\mathcal{O}$		$\mathcal{U}$		$\mathcal{S}$
$(A_{\mathcal{O}} = g_1^{o_1} g_2^{o_2},$ $B_{\mathcal{O}} = g_1^{o_3} g_2^{o_4})$		$(A_{\mathcal{U}} = g_1^{u_1} g_2^{u_2})$		
			$\xrightarrow{A, B, \mathrm{sign}(A, B)}$	
				$AB \stackrel{?}{\neq} 1$
			$\xleftarrow{c}$	$c \in \mathbb{Z}_q^*$
		$c \stackrel{?}{\neq} 1$		
	$\xleftarrow{c'}$	$c' \leftarrow \beta(1-c)/\alpha \bmod q$		
Coin unspent?				
$c' \stackrel{?}{\neq} 0$				
$r_1' \leftarrow o_1 + c' o_3 \bmod q$				
$r_2' \leftarrow o_2 + c' o_4 \bmod q$	$\xrightarrow{r_1', r_2'}$			
		$g_1^{r_1'} g_2^{r_2'} \stackrel{?}{=} A_{\mathcal{O}} B_{\mathcal{O}}^{c'}$		
		$r_1 \leftarrow \alpha(r_1' + u_1) \bmod q$		
		$r_2 \leftarrow \alpha(r_2' + u_2) \bmod q$		
		$r_3 \leftarrow \gamma + c(\alpha - \gamma) \bmod q$	$\xrightarrow{r_1, r_2, r_3}$	
				$g_1^{r_1} g_2^{r_2} g_3^{r_3} \stackrel{?}{=} AB^c$

FIGURE 17. The payment protocol for wallets with observers

As before, it follows from this that

$$\begin{aligned}
(c'r_1 - cr_1')/(c'-c) &= \alpha(o_1+u_1) + \beta o_3 \bmod q,\\
(c'r_2 - cr_2')/(c'-c) &= \alpha(o_2+u_2) + \beta o_4 \bmod q,\\
(c'r_3 - cr_3')/(c'-c) &= \gamma \bmod q,
\end{aligned}$$

$$\begin{aligned}
(r_1 - r_1')/(c-c') &= -\beta o_3 \bmod q,\\
(r_2 - r_2')/(c-c') &= -\beta o_4 \bmod q,\\
(r_3 - r_3')/(c-c') &= \alpha - \gamma \bmod q.
\end{aligned}$$

This in turn implies that the bank can compute

$$\begin{aligned}
\alpha(o_1+u_1) &= (c'r_1 - cr_1')/(c'-c) + (r_1 - r_1')/(c-c') \bmod q,\\
\alpha(o_2+u_2) &= (c'r_2 - cr_2')/(c'-c) + (r_2 - r_2')/(c-c') \bmod q,\\
\alpha &= (c'r_3 - cr_3')/(c'-c) + (r_3 - r_3')/(c-c') \bmod q,
\end{aligned}$$

and so

$$\begin{aligned}
o_1+u_1 = (\, r_1(c'+1) - r_1'(c+1))\,)/(\, r_3(c'-1) - r_3'(c+1)\,) \bmod q,\\
o_2+u_2 = (\, r_2(c'+1) - r_2'(c+1))\,)/(\, r_3(c'-1) - r_3'(c+1)\,) \bmod q.
\end{aligned}$$

That is, we have proven that if $\mathcal{U}$ followed both the withdrawal protocol and the payment protocol, the bank can compute I from the challenges and responses of two transcripts of the payment protocol.

16.8 Various remarks concerning this cash system

If the bank tries to frame the user, i.e. falsely accuses him of having double-spent the same information, the user can prove his innocence to a judge with a procedure that is merely a slight variation of that discussed in the two-party case. This procedure requires the observer to reveal his knowledge (o_1, o_2) of $A_{\mathcal{O}}$, and the bank must reveal the identifying information (v_1, v_2) he supposedly extracted. Since the observer does not know (u_1, u_2) of the user, it is easy to show that $(v_1 - o_1, v_2 - o_2)$ and (u_1, u_2) are different representations of $A_{\mathcal{U}}$ with respect to (g_1, g_2) with overwhelming probability if and only if the user did not double-spend (i.e. the bank tried to frame the user), and so an honest user can prove his innocence by e.g. coming up with $\log_{g_1} g_2$. That is, even if the observer would lie (which it could if $\mathcal{B}$ also built in $\log_{g_1} g_2$ within the observer, although this obviously would be very unwise), the probability of $\mathcal{U}$ not being able to prove his innocence is negligible.

In the full paper, we will prove that all the requirements for correctness are met. We here sketch the line of the proof. In the set-up phase, $\mathcal{S}$ accepts if and only if $\mathcal{U}$ is honest. Hence, I is such that $\mathcal{U}$ on its own does not know a representation. Now, during the withdrawal protocol, after $\mathcal{O}$ and $\mathcal{U}$ have proven knowledge of a representation of I, $\mathcal{O}$ sends to $\mathcal{U}$ a random number $B_{\mathcal{O}}$. This number will be used by $\mathcal{O}$ when the user wants to pay with the withdrawn information. Hence, it is not difficult to prove that $\mathcal{U}$ must make the splitting as dictated by the withdrawal protocol (possibly also using a term $g_1^{u_3} g_2^{u_4}$, but this does not alter anything as remarked earlier), since otherwise he will never be able to have the shop accept at payment time. If $\mathcal{U}$ follows the withdrawal protocol, it follows that the numbers A, B he ends up obtaining a signature on, are perfectly hidden from both $\mathcal{B}$ and $\mathcal{O}$. In the payment protocol, the fact that $B_{\mathcal{O}}$ is only used once also ensures that $\mathcal{U}$ will never be able to find out (o_1, o_2) from its responses (this is also why $\mathcal{O}$ must check that the challenge that

$\mathcal{U}$ sends him is not equal to 0). The transcript between $\mathcal{O}$ and $\mathcal{U}$ of valid executions of the payment protocol are easily shown to be statistically independent from those between $\mathcal{U}$ and $\mathcal{S}$, which is due to the blinding factors α, β of $\mathcal{U}$. Finally, we have already shown that the bank can extract a double-spender's identifying information, and hence we have indeed achieved the fact that even if a user can break the tamper-resistance, he will still be identified after double-spending, as in the basic two-party cash system.

Finally, observe that if the observer was broken by the user even before he established an account with the bank in the set-up phase, we have the correctness of the basic (two-party) cash system. In fact, the way the bank computes $(o_1 + u_1)$ and $(o_2 + u_2)$ is precisely the same. This is clearly an important reason why the cash system in the observer setting must resemble the basic cash system as closely as possible.

We end this section by remarking that one can adapt the withdrawal protocol and divert the payment protocol for the extension of our system to checks and divisibility in a similar manner. We believe however that these extensions in the observer setting can be achieved probably more efficiently, althought we have not investigated this yet.

17 Analogues in RSA-groups

As we discussed, the representation problem is also computationally difficult in RSA-groups. More specifically, there are a few variations in case the input group is always an RSA-group. In the descriptions of these three variations, n is the product of two (or more) distinct primes.

- Given k distinct elements $(g_1, \ldots, g_k)$ and h, find a representation $(a_1, \ldots, a_k)$ of h such that $\prod_{i=1}^{k} g_i^{a_i} = h \bmod n$. As we mentioned, this problem is mentioned in [6], and it is stated to be equivalent in computational difficulty to factoring.
- Given k distinct exponents $(v_1, \ldots, v_k)$ and h, find a representation $(X_1, \ldots, X_k)$ such that $\prod_{i=1}^{k} X_i^{v_i} = h \bmod n$. As far as we know, this problem is equivalent in computational difficulty to computing RSA-roots (see [19] for a discussion that is of interest in this matter).
- This is a combination of the previous two variations. Given l distinct exponents $(v_1, \ldots, v_l)$, $k - l$ distinct elements $(g_{l+1}, \ldots, g_k)$ and h, find a representation

  $$(X_1, \ldots, X_l, a_{l+1}, \ldots, a_k)$$

  such that $\prod_{i=1}^{l} X_i^{v_i} \prod_{i=l+1}^{k} g_i^{a_i} = h \bmod n$. The problem with $l = 1$, $k = 2$ has been used in [25, 32] and is equivalent to factoring.

For a similar result to Corollary 8 to hold, the elements v_i must be pairwise co-prime.

We remark that in [25, 32] a scheme is discussed that is suitable for the payment protocol, since one of the responses (but not both!) is linear in the challenge. For our viewpoint the other response (which is not a line) can be viewed as necessary to guarantee the self-certifiability property. Furthermore, the splitting technique can be applied here as well. It is easy to see that almost all tools we used in our cash system can be adapted to all three variations, except unfortunately the blind signature scheme. That is, we have not yet been able to find a restrictive blind signature scheme in RSA.

We further remark that in our cash system, we can use a group G_q with the order of the group q unknown to the participants in the system other than the bank. To this end, the bank can for example generate a prime p such that $p-1=qr$, with q,r primes of approximately equal length, and take G_q to be the subgroup of Z_p^* of order q. One can then apply the technique of [5] in the payment protocol. This consists of the responses being taken modulo $p-1$ instead of modulo q. In order to attack the payment protocol (not using the withdrawal protocol), one is faced with a more difficult task since it also requires the ability to factorize. Nevertheless, the correctness of this slight variation of our system still hinges on the Diffie-Hellman assumption.

18 The Decision Diffie-Hellman problem and other open problems

In this final section, we discuss a few open problems related to our cash system. The first of these is of more general importance, and we hence devote a separate paragraph to it. We further remark that in a realistic implementation of our system, all kinds of signatures (receipts proving that a certain action occurred) must be included. Since this is easy to incorporate and falls outside the scope of the cryptographic model, we have not discussed this.

The Decision Diffie-Hellman problem It is conceivable that there may be blind signature schemes in groups of prime order that need somewhat less space and verification computations than the one we use. In our system, there is one unspecified hash-function, which we would like to get rid of (in a nice way, that is) since that would probably increase the degree of provability of correctness even more. Currently known digital signature schemes in a discrete log setting (i.c. $\mathbb{Z}_p^*$ or G_q), such as described e.g. in [5, 36], other than that of [15] are all similar in the sense that they use the hash-function technique to turn a proof of knowledge into a signature scheme, and cannot be converted to blind signature schemes that have the property of restricted blinding manipulations. It would obviously be beneficial to the degree of provability of correctness of our system if there were a digital signature scheme suitable for $\mathbb{Z}_p^*$ that does not contain an unspecified hash-function (such as extracting v-th roots in RSA), and that can be converted easily into a blind signature protocol. Ideally, it would have the property of restrictive blinding.

From this point of view, it would be beneficial for our cash system (and for discrete log based cryptography in general) if the undeniable signature of [9, 10] were actually a digital signature. To this end, we here observe that there is an intimate relation between this undeniable signature and the Diffie-Hellman problem, which as far as we know has been overlooked in literature. We state this relation in the form of a decision problem:

Definition 21 *The Decision Diffie-Hellman problem is the problem of deciding whether, given inputs g, g_1, g_2, and h, the latter is the Diffie-Hellman key of g_1 and g_2 with respect to g. An algorithm is said to solve this problem to the base g if, for randomly chosen inputs g_1, g_2 and h, it decides correctly with nonnegligible probability. The Decision Diffie-Hellman assumption states that, for all $g \neq 1$, there is no such polynomial-time algorithm.*

With g^x being the signer's public key, m a message and m^x the undeniable signature on m, the Diffie-Hellman problem is equivalent to the problem of forging undeniable signatures on given messages, and the problem of being able to determine without help of the signer whether an undeniable signature is correct is equivalent to the Decision Diffie-Hellman problem. If the Decision Diffie-Hellman assumption is false, then the undeniable signatures are in fact digital signatures.

We here remark that the same problem has been overlooked in the cash system of [22]: linking withdrawals to payments in that system is at most as difficult as breaking the Decision Diffie-Hellman problem.

At the same time, if the Decision Diffie-Hellman assumption turns out to be false then one has as an important consequence the fact that a proposition similar to Proposition 2 is true for the Diffie-Hellman problem, which would obviously greatly increase our believe in the Diffie-Hellman assumption. More importantly, it would imply that, as in RSA, there is a trapdoor one-way function in discrete logarithms, and so in general this would probably allow the construction of many cryptographic tools based on discrete logarithms analogue to those in RSA-groups.

Note that not only is it of interest to know whether the Decision Diffie-Hellman problem is in the complexity class $\mathcal{P}$, but even whether it is in $\mathcal{R}$ or in $\mathcal{BPP}$. In the case latter cases, the unusual situation arises that the undeniable signatures in literature are digital signatures that are probabilistically recognizable, which is probably just as good for all practical purposes.

Other open problems We list some other open problems, some of which have already been mentioned earlier in the text:

- Do there exist suitable restrictive blind signature schemes in RSA-groups? We could then build cash systems in RSA-groups in a similar way to the one descibed in this paper.
- Is there a more efficient restrictive blind signature scheme in G_q?
- Can the extensions be achieved in the setting of wallets with observers in a more efficient way, under the requirement of no shared information?
- In our extensions to multi-show cash and divisibility, there is full linkability. Can one achieve the extensions without linkability?

19 Acknowledgements

The technical report lying before you is the result of research that took place over the past half year. I owe many thanks to my colleagues Ronald Cramer, Niels Ferguson and Berry Schoenmakers at CWI, and Torben Pedersen at MI Aarhus for inspiring discussions while this work was in progress. I am in particular grateful to Ronald for discussions about the basic payment protocol, and to Torben for pointing out that my earlier proof of Proposition 12 was incorrect, as well as for his help with the batched confirmation protocol of Section 20. The way in which Niels Ferguson, concurrently working on an off-line electronic cash system based on RSA, avoided the cut-and-choose methodology in the payment protocol inspired me to try to achieve this property in a discrete logarithm environment. Recently, Niels and I found out that Franklin and Yung in their article also apply this technique, although they

still use the cut-and-choose methodology in the withdrawal protocol. The fact that there are hence currently three different cash systems that all have such an efficient payment protocol to me indicates that secure privacy-protecting electronic cash systems are rapidly becoming efficiently realizable.

I especially want to thank David Chaum. This work would never have existed at all if it weren't for his innovative work, giving rise to the exciting field of privacy-protecting electronic cash, and the stimulating working environment he created for me at CWI's cryptogroup.

I want to end this report by remarking that only a few sections of this technical report have been read by others than myself. Especially, none of the extensions has been read yet. Although I feel pretty confident about the correctness of the system, it clearly is necessary that it is studied by other people as well. I therefore encourage you to try to break the system. If you find any flaws whatsoever, I would be pleased to be informed about them. You can contact me by e-mail.

References

[1] Antwerpen, H. van, “Electronic Cash”, Centre for Mathematics and Computer Science, Amsterdam (1990).

[2] Bos, J. and Purdy, G., “A voting scheme”, Rump session of Crypto ’88 (does not appear in the proceedings).

[3] Brands, S. and Chaum, D., “How to prevent the mafia-fraud by using distance-bounding protocols”, Tech. Rep., C.W.I. (to appear). An abstract was submitted to Eurocrypt ’93.

[4] Brands, S., Chaum, D., Cramer, C., Ferguson, N. and Pedersen, T., “Transaction systems with observers”, Tech. Rep., C.W.I., (to appear).

[5] Brickell, E. and McCurley, K., ‘An interactive identification scheme based on discrete logarithms and factoring’, Journal of Cryptology, Vol. 5, no. 1 (1992), pages 29–39.

[6] Chaum, D., Evertse, E. and Graaf, J. van der, “An improved protocol for demonstrating possession of discrete logarithms and some generalizations”, Eurocrypt ’87, LNCS 304, Springer-Verlag, pages 127–141.

[7] Chaum, D., Boer, B. den, Heyst, E. van, Mjolsnes, S. and Steenbeek, A., “Efficient off-line electronic checks”, Eurocrypt ’89, LNCS 434, Springer-Verlag, pages 294–301.

[8] Chaum, D., Fiat, A. and Naor, M., “Untraceable Electronic Cash”, Crypto ’88, LNCS 403, Springer-Verlag, pages 319–327.

[9] Chaum, D. and Antwerpen, H. van, “Undeniable Signatures”, Crypto ’89, LNCS 435, pages 212–216.

[10] Chaum, D., “Zero-knowledge undeniable signatures”, Eurocrypt ’90, LNCS 473, Springer-Verlag, pages 458-464.

[11] Chaum, D., “Blind signatures for untraceable payments”, Crypto ’82, Springer-Verlag, pages 199–203.

[12] Chaum, “Achieving electronic privacy”, Scientific American, Aug. 1992, pages 96–101.

[13] Chaum, D., Heijst, E. van, and Pfitzmann, B., “Cryptographically Strong Undeniable Signature, Unconditionally Secure for the Signer”, Crypto ’91, LNCS 576, pages 470–484.

[14] Chaum, D. and Pedersen, T., “Transferred money grows in size”, presented at Eurocrypt ’92.

[15] Chaum, D. and Pedersen, T., “Wallet databases with observers”, Preproceeding of Crypto ’92, pages 3.1–3.6.

[16] Coster, M., “ Some algorithms on addition chains and their complexity”, TR CS-R9024, Centrum voor Wiskunde en Informatica, Amsterdam (June 1990).

[17] Cramer, R. and Pedersen, T., “Improved privacy in wallets with observers’, submitted to EuroCrypt 1993.

[18] Diffie, W. and Hellman, M., "New Directions in Cryptography", IEEE Transactions on Information Theory 22/6 (1976), pages 644–654.

[19] Evertse, J. and Van Heijst, E., "Which new RSA-signatures can be computed from given RSA-signatures?", Journal of Cryptology, Vol. 5, no. 1 (1992), pages 41–52.

[20] Feige, U., Fiat, A. and Shamir, A., "Zero-knowledge proofs of identity", Journal of Cryptology 1 (1988), pages 77–94.

[21] Fiat, A. and Shamir, A., "How to prove yourself: practical solutions to identification and signature problems", Crypto '86, Springer-Verlag, (1987), pages 186–194.

[22] Franklin, M. and Yung, M., "Towards provably secure efficient electronic cash", Columbia Univ., Dept of Comp. Sc., TR CUCS-018-92, April 24, 1992.

[23] Hayes, B., "Anonymous one-time signatures and flexible untraceable electronic cash", Auscrypt '90, LNCS 453, pages 294–305.

[24] Heijst, E. van, and Pedersen, T., "How to Make Efficient Fail-stop Signatures", Eurocrypt '92, Extended Abstracts, 24-28 Mai 1992, Balatonfüred, Hungary, pages 337–346.

[25] Heijst, E. van, Pedersen, T. and Pfitzmann, B., "New constructions of fail-stop signatures and lower bounds", Crypto '92, pages 1.9–1.14.

[26] Knuth, D.E., *Seminumerical Algorithms*, Volume 1 of *The Art of Computer Programming*, Addison-Wesley (1968). Second edition (1973).

[27] McCurley, K., "The discrete logarithm problem", AMS Proc. Symp. Appl. Math, Vol. 42: Cryptology and Computational number theory (1991), pages 49–74.

[28] Olivos, J., "On vectorial addition chains", Journal of Algorithms 2 (June 1981), pages 13–21.

[29] Okamoto, T. and Ohta, K., "Disposable Zero-knowledge authentications and their applications to untraceable electronic cash", Crypto '89, LNCS 435, Springer-Verlag, Heidelberg 1992, pages 481–496.

[30] Okamoto, T. and Ohta, K., "Universal electronic cash", Crypto '91, LNCS 576, Springer-Verlag, Berlin 1992, pages 324–337.

[31] Okamoto,T. and Ohta,K., "Divertible zero knowledge interactive proofs and commutative random self-reducibility", Eurocrypt '89, Springer-Verlag, pages 134-149.

[32] Okamoto, T., "Provably Secure and Practical Identification Schemes and Corresponding Signature Schemes", Proceeding of Crypto '92, pages (1-15) – (1-25).

[33] Pedersen, T., "Non-interactive and information-theoretic secure verifiable secret sharing", CRYPTO '91, LNCS576, pages 129–140.

[34] Pedersen, T., "Distributed provers and verifiable secret sharing based on the discrete logarithm problem", Ph. D. Thesis, DAIMI PB - 388 (March 1992), Computer Science Dept. Aarhus University, Denmark.

[35] Santis, A. de, and Persiano, G., "Communication efficient zero-knowledge proofs of knowledge (with applications to electronic cash)", STACS '92, LNCS 577, pages 449–460.

[36] Schnorr, C.P., "Efficient Signature Generation by Smart Cards", Journal of Cryptology, Vol. 4, No. 3, (1991), pages 161–174.

[37] Shamir, A., "How to share a secret", Communications of the ACM 22 (1979), pages 612–613.

Batching the Confirmation Protocol of Undeniable Signatures. In an undeniable signature scheme there are two basic types of protocol, being a confirmation protocol and a disavowal protocol. In the confirmation protocol for the undeniable signatures of [1, 9], the signer (with public key (g, g^x)) proves to a verifier that the signature z he obtained on a message m is such that $\log_m z = \log_g h$. By the Decision Diffie-Hellman assumption, he cannot feasibly determine this himself.

There are two zero-knowledge versions of the confirmation protocol known in literature, one ([34]) being a slight variation of the original ([10]). The protocol we show how to batch here is, although the choice is arbitrary, the former. In the protocol, $\mathcal{V}$ has polynomial-time computing power and $\mathcal{P}$ unlimited computing power.

We show here how the confirmation protocol can be batched such that the signer convinces the receiver of the correctness of polynomially (i.e. $k = O(|p|^n)$ for some $n \in \mathbb{N}$) many undeniable signatures $z_1, \ldots, z_{k-1}$ on messages $m_1, \ldots, m_{k-1}$. Before the protocol takes place, $\mathcal{V}$ has received $k-1$ numbers $z_1, \ldots, z_{k-1}$, supposedly constructed by $\mathcal{P}$ from $m_1, \ldots, m_{k-1}$ by raising each to his secret key x. For clarity, this "initialization step" is also depicted in Figure 18, together with the actual protocol itself. This consists of the following steps:

Step 1. $\mathcal{V}$ generates at random an index-tuple $(a_1, \ldots, a_k)$, computes $h = (\prod_{i=1}^{k-1} m_i^{a_i})\, g^{a_k}$, and sends the result to $\mathcal{P}$.

Step 2. $\mathcal{P}$ generates a number $c \in_{\mathcal{R}} \mathbb{Z}_q$ at random, computes $A_0 = h^c$ and $A_1 = h^{cx}$, and sends these two numbers to $\mathcal{V}$.

Step 3. $\mathcal{V}$ sends his representation $(a_1, \ldots, a_k)$ of h with respect to $(m_1, \ldots, m_{k-1}, g)$ to $\mathcal{P}$.

Step 4. $\mathcal{P}$ verifies that $h = (\prod_{=1}^{k-1} m_i^{a_i})\, g^{a_k}$. If the verification holds, he sends c to $\mathcal{V}$.

Step 5. $\mathcal{V}$ verifies that $A_0 = h^c$ and $A_1 = ((\prod_{i=1}^{k-1} z_i^{a_i})\,(g^x)^{a_k})^c$, and accepts if and only if both verifications hold.

$\mathcal{P}$ (whom we allowed to have unlimited computing power) is said to cheat in this protocol if he can have $\mathcal{V}$ accept whereas at least one of the signatures z_i is incorrect.

Proposition 22 *If $\mathcal{V}$ follows the protocol, $\mathcal{P}$ cannot cheat with nonnegligible probability of success.*

Proof. Writing all numbers to the base g (for $g \neq 1$), we denote A_0 by g^{c_0}, A_1 by g^{c_1}, and, (for all $i \geq 1$), $\log_g m_i$ by l_i, and $\log_{g_i} z_i$ by s_i. We then see that an honest $\mathcal{V}$ accepts if and only if

$$(g^{l_1 a_1} g^{l_2 a_2} \cdots g^{l_{k-1} a_{k-1}} g^{a_k})^c = g^{c_0} \text{ and } (g^{l_1 s_1 a_1} g^{l_2 s_2 a_2} \cdots g^{l_{k-1} s_{k-1} a_{k-1}} g^{x a_k})^c = g^{c_1}.$$

That is, if the honest $\mathcal{V}$ accepts, the following relations must hold:

$$\overbrace{\begin{pmatrix} l_1 & \cdots & l_{k-1} & 1 \\ l_1 & \cdots & l_{k-1} & 1 \\ l_1 s_1 & \cdots & l_{k-1} s_{k-1} & x \end{pmatrix}}^{\mathbf{M}} \overbrace{\begin{pmatrix} a_1 \\ \vdots \\ a_k \end{pmatrix}}^{\mathbf{a}} = \overbrace{\begin{pmatrix} \log_g h \\ c_0/c \\ c_1/c \end{pmatrix}}^{\mathbf{c}} \bmod q.$$

Observe that if $s_i = x$ for all $i \in \{1, \ldots, k-1\}$, then $\text{rank}(\mathbf{M}) = 1$. Hence there is a solution if and only if $c_0/c = \log_g h \bmod q$ and $c_1/c = x \log_g h \bmod q$, and because the arithmetic is in a field, there are exactly q^{k-1} solutions $\mathbf{a}$ of $\mathbf{Ma}{=}\mathbf{c}$. These are precisely the representing index-tuples of h with respect to $(m_1, \ldots, m_k)$.

If, on the other hand, not all s_i are the same ($\mathcal{P}$ is cheating), then $\text{rank}(\mathbf{M}) = 2$. With $c_0/c = \log_g h \bmod q$ and $c_0 \neq c_1$ (otherwise, there is no solution at all), there are q^{k-2} solutions $\mathbf{a}$. These solutions constitute a negligible fraction $1/q$ of all representing index-tuples of h. Because $\mathcal{V}$ follows the protocol, each of the representing index-tuples of h has equal probability of having been chosen by $\mathcal{V}$ in Step 1, so $\mathcal{P}$ has no better cheating strategy than hoping that $\mathcal{V}$ chooses his representing index-tuple of h in the subset of representing index-tuple that are solutions $\mathbf{Ma} = \mathbf{c}$. The probability of successful cheating is hence at most $1/q$. $\square$

Since k is assumed to be polynomial in the length of the input, it follows straightforward from the proof of zero-knowledgeness in [34] that this batched protocol is still zero-knowledge.

Prover $\mathcal{P}$ (Unbounded)		Verifier $\mathcal{V}$ (Poly-time)
(g, g^x)		
	$\xleftarrow{m_1, \ldots, m_{k-1}}$	
$z_i \leftarrow m_i^x$	$\xrightarrow{z_1, \ldots, z_{k-1}}$	
	$\vdots$	
		$a_1, \ldots, a_k \in_{\mathcal{R}} \mathbb{Z}_q$
	$\xleftarrow{h}$	$h \leftarrow (\prod_{i=1}^{k-1} m_i^{a_i})\, g^{a_k}$
$c \in_{\mathcal{R}} \mathbb{Z}_q$		
$A_0 \leftarrow h^c$		
$A_1 \leftarrow h^{cx}$	$\xrightarrow{A_0, A_1}$	
	$\xleftarrow{a_1, \ldots, a_k}$	
$h \stackrel{?}{=} (\prod_{i=1}^{k-1} m_i^{a_i})\, g^{a_k}$	$\xrightarrow{c}$	
		$A_0 \stackrel{?}{=} h^c$
		$A_1 \stackrel{?}{=} ((\prod_{i=1}^{k-1} z_i^{a_i})(g^x)^{a_k})^c x$

FIGURE 18. The batched confirmation protocol of undeniable signatures