Self-Delegation with Controlled Propagation – or – What If You Lose Your Laptop

Oded Goldreich* Birgit Pfitzmann† Ronald L. Rivest‡

September 1997

Abstract

We introduce delegation schemes wherein a user may delegate rights to *himself*, i.e., to other public keys he owns, but may not safely delegate those rights to *others*, i.e., to their public keys. In our motivating application, a user has a primary (long-term) key that receives rights, such as access privileges, that may not be delegated to others, yet the user may reasonably wish to delegate these rights to new secondary (short-term) keys he creates to use on his laptop when traveling, to avoid having to store his primary secret key on the vulnerable laptop.

We propose several cryptographic schemes, both generic and practical, that allow such self-delegation while providing strong motivation for the user not to delegate rights that he only obtained for personal use to other parties.

Keywords: Delegation, subkeys, key hierarchy, Zero-Knowledge Proofs, Knowledge Complexity, Threshold Schemes, gradual release of secrets, Signature Schemes, Non-Interactive Zero-Knowledge, Commitment Schemes.

*Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, ISRAEL. E-mail: oded@wisdom.weizmann.ac.il. Currently visiting LCS, MIT. Partially supported by DARPA grant DABT63-96-C-0018.

†Universität des Saarlandes, Fachbereich Informatik, D-66123 Saarbrücken, GERMANY. Email: pfitzmann@cs.uni-sb.de. Work partially done while visiting MIT and partially at the University of Hildesheim. Supported by DFG (German Research Foundation).

‡Laboratory for Computer Science, MIT, Cambridge, Mass., USA. Email: rivest@theory.lcs.mit.edu. Supported by DARPA grant DABT63-96-C-0018.

1 Introduction

In a networked world, users are represented by their public keys. A message whose digital signature verifies with a user's public key is assumed to have been signed by that user. This assumption requires that users maintain the secrecy of their signing keys. In real life, however, users may wish to use relatively insecure computing devices, e.g. networked UNIX machines or laptops, to effect digital transactions. How can a user make use of such insecure devices while preserving the secrecy of his secret key?

One natural solution is for the user to use the secret key of his *primary* (long-term) key pair *only* on secure computing devices, and to create a *secondary* (temporary, short-term) key pair as needed for use on an insecure computing device. The user then delegates authority from his primary key pair to his secondary key pair by signing a "delegation certificate" of the form, "On dates [dates] any signatures verifiable with [secondary public key] may be treated as if they were verifiable with [primary public key]. [Signed with primary secret key]." The user keeps his primary secret key in a safe place, and uses only his secondary secret key and his delegation certificate on the insecure computing device. When signing a message with the secondary key, the user attaches the delegation certificate so that the message recipient may verify both the message signature and the delegation of signing authority from primary key pair to secondary key pair. If the secondary secret key is compromised, loss is minimized because the delegation certificate expires quickly.

Such a delegation scheme from primary to secondary keys is very natural and flexible, and seems to solve our initial problem. Certain extensions come easily to mind, e.g., the delegation certificate may limit the delegated authority to certain applications or to transactions of restricted value. Related schemes are well-known in the literature, e.g., [11] where primary keys are held on smartcards and authority is temporarily delegated to secondary keys on work stations.

Such use of delegation certificates, however, is in conflict with the interests of service providers to prevent the *propagation* of certain rights and benefits. For example, suppose a web site that should only be accessed by kids that subscribe to a certain magazine. This policy is enforced by requiring subscribers to sign access requests with their public key. The web site maintains an access control list specifying the public keys of all subscribers or gives each subscriber a certificate asserting his subscriberhood. This works fine, *unless* user-generated delegation certificates are also allowed. With unrestricted use of such delegation certificates, a mischievous kid can delegate his access authority to all of his friends, by signing delegation certificates that authorize their public keys to have the access rights that his public key has been granted.

Resolving this conflict is the central goal of this paper. We ask: *How can delegation of rights be managed such that a user can delegate these rights to himself, but not to others?*

It may seem that the conflict is unresolvable, and indeed to some extent it is: Even if no delegation is allowed, a user can simply give away his *primary* secret key. Of course, tamper-resistant hardware can restrict (and in combination with biometrics even prevent) such acts, but our primary concern is the use of *insecure* hardware. Then, such access control is typically based on the assumption that a user is highly motivated *not* to give away his primary (long-term) secret key. A user's primary key is the fundamental representation of his network identity, and a user who gives his secret key away may be giving up his bank account, voting rights, etc. The *new* problem introduced by simple delegation is that such motivation is not given for secondary secret keys, in particular if the delegation certificate limits the authority, i.e., our kid can allow his friends to access the web site without enabling them to access his digital pocket money.

Thus, we propose to deter users from delegating personal rights to other users by ensuring that the recipient must learn something about the primary secret key of the delegator. A user will then again be highly motivated *not* to delegate such rights so as not to risk giving up the fundamental rights associated with the primary secret key. All of our suggestions are based on this assumption.

The basic idea of all our suggestions is that the user has to *validate* his secondary keys with respect to his primary secret key (in addition to signing a delegation certificate as above). Access requests signed with a secondary key will only be granted for secondary keys validated in this way. Our proposals differ in the techniques used to encode the information about the delegator's primary secret key in the secret validation information and to allow service providers to verify the presence of this information without learning it. It seems, however, that the following two goals are still in conflict:

1. Restricted damage from key disclosure: Minimize the damage caused by the inadvertent loss or theft of the user's current validated secret secondary key.

2. Controlled propagation: Discourage a user from delegating certain rights to others by ensuring that the recipient learns something about the user's primary secret key.

The conflict arises because the thief in the first goal is essentially indistinguishable from the recipient in the second goal. Our approach is to try and strike a balance between these conflicting requirements:

1. Loss of a *few* validated secondary secret keys will not endanger the security of the primary secret key. That is, they yield little or no information or knowledge about the primary secret key. Anyone who steals the secondary secret key of a user, even a few times, is therefore restricted to exactly the power the user delegated to this secondary key.

2. *Many* validated secondary secret keys allow the recipient to recover the primary secret key efficiently. Thus, users are discouraged from distributing secondary keys to other parties on a large scale, because the others would gain power beyond what the user wanted to delegate to them with these secondary keys.

We present several constructions achieving such a balance. They are categorized by the following parameters:

- Generic vs. specific: In Section 3, we present generic schemes which apply to any primary and secondary keys. These scheme rely on non-interactive zero-knowledge proofs [4] and thus on the existence of trapdoor one-way permutations [7, 16]. In Section 4, we present more specific and efficient constructions where the primary secret key pair must belong to a discrete-logarithm based cryptographic scheme like Schnorr or DSS signatures [19, 6].

- Gradual vs. threshold: In Constructions 1 and 2, the security of the primary secret key degrades gradually with the number of secondary secret keys available to an adversary until the primary secret key is totally revealed. In Constructions 3 to 5, the primary secret key is fully secret up to a certain number of secondary secret keys available to an adversary, but is easily recovered from any larger number of secondary secret keys.

  Additionally, some schemes allow the application to freely fix after how many secondary secret keys, e.g., 3, 10, or 100, the adversary will get the primary secret key (i.e., the slope of the gradual degradation or the threshold, respectively), whereas in others this number is predefined by other system parameters, e.g., the key length.

- Limitations for secondary keys: We handle the limitations, e.g., to a time period, a dollar amount limit, or to certain transaction types, associated with the delegation to a secondary key in a general way. We call a description of these limits a *limitations index* for a secondary key associated with a particular primary key. A relevant question for our schemes is whether the domain of potential limitation indices is "huge" or "relatively small". For example, if secondary keys are simply associated with days, the domain is relatively small, whereas if a secondary key may need to be generated for any possible 1000-words English-text document, the domain is huge. Constructions 1 and 4 are for unbounded domains, Constructions 2, 3 and 5 for bounded domains. Among the latter, the bounds in Constructions 3 and 5 are of little practical significance, whereas the bound in Construction 2 is indeed severe. Specifically, in Constructions 3 and 5 the domain of potential limitation indices is exponential in the security parameter (i.e., the length of the key), whereas in Construction 2 the domain is exponential in the slope/threshold parameter mentioned above.

Note that of course we do not claim that all delegation should only be self-delegation – in many applications, delegation to others is explicitly desired, e.g., to support organizational hierarchies, or at the discretion of the "user", e.g., the right to spend his own money or access rights to a certain *amount* of service. Our proposals are not in conflict with this: Both types of rights can be given on the same primary and secondary keys and even with the same delegation certificates. The only difference is that the presence of validation information related to the key of the *original* owner of the right will be verified for rights that should only be self-delegated, whereas other rights can freely be delegated to arbitrary public keys.

2 The Model

Our model and constructions of self-delegation schemes only concern the core part of the applications described so far and are therefore largely independent of the concrete application. In particular, we only deal with the primary and secondary keys, and not with certificates that third parties give to assign rights to these keys, such as the above-mentioned certificate asserting subscriberhood – these can be realized in standard ways by signatures with keys held by the third parties *on* the keys handled in the self-delegation scheme, and thus be added to our constructions in a modular way.

Furthermore, we do not explicitly consider the user's own authorization of the secondary keys, i.e., the delegation certificates as described at the beginning. This is not quite as independent of our schemes as the statements by third parties, but whenever the underlying cryptographic scheme is a signature scheme, the delegation certificates can be made in a standard way with the primary secret key.

The notion of a generic scheme will more precisely mean the following: An *arbitrary* underlying cryptographic scheme may be given that contains an algorithm for generating primary key pairs and one for generating secondary key pairs, possibly depending on a primary key pair. The typical case is simply that an algorithm *gen* for generating *one* key pair is given, and all primary and secondary key pairs are generated independently with this algorithm. Our mechanism allows to control propagation of such secondary keys without posing additional dangers on the underlying scheme and without making assumptions about how the keys of the underlying scheme may be used. For instance, signing (with a secret key) is not a zero-knowledge procedure, and in some protocols secret keys may even sometimes be revealed. Our controlled self-delegation schemes should not pose any *additional* risk to the keys, i.e., beyond the risk involved in using the given underlying cryptographic schemes. The schemes in Section 3 are of this type, and those in Section 4 are only slightly less generic in that they assume a specific generation algorithm for the primary secret key. We briefly mention variants that may be a little more efficient, but require that primary and secondary keys are only used in well-defined ways, e.g., within a specific signature scheme. In the rest of the model section, we therefore concentrate on generic schemes.

We have to consider three parties: a server, which corresponds to a key certification infrastructure, users, who have primary and secondary keys, and verifiers, with whom the users interact using their secondary keys. A self-delegation scheme with controlled propagation has four protocols:

- Server setup: Here the server generates and publishes global parameters.
- Registration: Here the user has generated a primary key pair (sk, pk) from an underlying cryptographic scheme and registers the primary public key with the server. The server maintains a public file with a public record for each user, which will contain pk and possibly additional information. We call all the secret information the user stores at the end of registration his extended primary secret key (and often omit "extended" and call it sk where the distinction is not important; similarly with pk and the public record).
- Secondary key validation: Here the user has generated a secondary key pair (sk_l, pk_l) from the underlying cryptographic scheme, which he wants to validate for a certain *limitations index* l. He produces a validation tag val_l that will later, when he no longer has the primary secret key available, allow him to convince verifiers of the validity of the secondary key pair.
- Verification: Here a user wants to convince a verifier, typically a service provider, that a certain public key pk_l can be used as a secondary key for the primary public key pk.

  The verifier also inputs the limitations index l for which pk_l is supposed to be valid and global parameters published in server setup. Note that controlled propagation would make no sense if the user could set l without l being checked by the verifier; for example, if the secondary key is limited to a specific date, then it is important that the verifier enters the current date rather than let the alleged user enter a date of his choice (which would always be the one for which the key is valid).

  We stress that the user's secret inputs are only sk_l and val_l; he now no longer has the primary secret key available.

These protocols must be efficiently computable, i.e., at least in probabilistic polynomial-time, and in time comparable with the underlying cryptographic schemes for practical schemes. Furthermore, correctly validated secondary keys for correctly registered primary public keys should (almost) always be accepted by honest verifiers.

We call a triple (sk_l, val_l, pk_l) a correctly validated secondary key triple if it is a possible output of the correct secondary key generation and validation program. This is what a thief gets from an honest user. However, dishonest users who try to give away secondary keys to others are not restricted to correctly validated keys. We say that a user gives a second person a valid secondary key triple (for a public key pk and a limitations index l) if the information enables the second person to convince an honest verifier almost certainly. Formally, this notion is of course only meaningful in connection with the second person's program, which may be arbitrary, yet must be efficient. For simplicity, we currently only consider giving away full-quality secondary keys, in contrast to information that, say, convinces the verifier with probability 1/2.

Our two goals can now be formulated in more detail as follows:

- Restricted damage from key disclosure: If a certain user is honest, it should be infeasible for an adversary (thief) who only obtains *few* correctly validated secondary key triples to generate additional valid secondary key triples. Here, "additional" means for a limitations index l^* for which the adversary has not obtained a correctly validated secondary key triple. The adversary should also not get too much knowledge about existing non-stolen secondary secret keys sk_l nor about the primary secret key. If these keys are only used for a specific purpose, e.g., signing, it is sufficient to define that the adversary cannot forge in the underlying scheme. In a generic scheme, we have to speak about the knowledge gained about these keys, or rather any knowledge gained by validation tags; see the discussion before Proposition 1.

  More precisely, gradual schemes have a parameter ℓ so that each correctly validated secondary key triple only gives ℓ bits of knowledge. Schemes with threshold have a parameter k so that less than k correctly validated secondary key triples do not give the adversary any knowledge.

- Controlled propagation: If a user gives a second person *many* different valid secondary key triples, then the second person can efficiently compute the user's primary secret key. Here, "different" means for different limitations indices.

  In schemes with threshold, "many" typically means at least k, but there are also schemes with two different thresholds, i.e., a certain gap. Gradual schemes have a similar parameter b, which will typically be about n/ℓ, where n is the key length and ℓ is as in the first goal.

For simplicity, we assume that all the security parameters ℓ, k etc. are chosen in server setup. Typically, users are concerned with restricting the damage caused by disclosed secondary keys and therefore wish parameters such as k to be large. On the other hand, service providers, mainly concerned with propagation control, want parameters such as k to be small.

We allow the following types of attacks:

- The adversary for the first goal, "restricted damage", knows the public record and includes both other users and verifiers who have verified some of this user's secondary public keys and with whom the user has then interacted using these keys, e.g., by signing messages.

  In a strong form of the requirement, the adversary also includes the server, i.e., the user need not trust the server. This is a realistic requirement if we consider signature schemes, where the server should not learn the users' secret keys.

- The adversary for the second goal, "controlled propagation", consists of any number of colluding users and other verifiers. Typically, we only speak of "the user" because independence is clear. For some constructions, we do not restrict the user to polynomial-time computation for this requirement, but rather assume that he will only get a certain (large) number m of answers from the server (see Proposition 1).

In all the following schemes, the first goal, "restricted damage", is stated and proven with respect to a static adversary, i.e., the limitations indices of the revealed validation tags are determined beforehand rather than chosen adaptively by the adversary. We believe that all schemes are in fact adaptively secure, but refrain at this point from formulating and proving our belief. We also assume for the time being that the user only carries out one protocol at a time, i.e., we do not consider interleaving of protocols.

3 Generic Schemes

The basic idea to achieve our two goals is to augment each secondary secret key by some little "piece of information" σ_l about the primary secret key sk. This will be part of the validation tag val_l, and the holder of a secondary key will only be able to convince verifiers if he *knows* this σ_l. The process of convincing the verifier will be done in zero-knowledge, so that verifiers do not gain an advantage in finding a user's primary secret key. (Recall that only people to whom the user gives, in violation of the controlled propagation requirement, many secondary *secret* keys should be able to find the user's primary secret key.) Thus, what the user has to put into σ_l is related to the gradual release of a secret, as introduced for secret exchange in [3], with the difference that in our case the people who get the pieces of information and those who verify their correctness are different and do not trust each other.

To make it impossible for the user to give the *same* information with every secondary key he passes to others, the precise value of σ_l is determined by the limitations index l. In particular, suppose we manage to force the user to give away, with each secondary key, an independent random linear combination of the bits of the primary secret key sk. Then, with overwhelmingly high probability, a second person who obtained $2n$ secondary keys will be able to derive the entire n-bit primary secret key.

In the proof that he knows σ_l, the user also has to prove that σ_l is correct with respect to the primary secret key sk, which is no longer available in the verification stage. Hence this part of the proof will be precomputed in the secondary key validation protocol. We use a non-interactive zero-knowledge proof system [4] for this. In the following, we assume that we are given such a proof system where one common random string Ref can be used for an arbitrary number of proofs [7]. (This result also applies to the more efficient basic construction in [16].) In schemes with a small bounded set of limitations indices, we could improve efficiency by using a non-interactive zero-knowledge system without this property, i.e., capable of only proving one assertion, and providing enough reference strings in advance.

Finally, we have to guarantee that all σ_l's refer to the *same* primary secret key sk. This is easiest if the underlying cryptographic scheme guarantees that for every primary public key pk, there is only one possible primary secret key sk. This property is fulfilled in quite a number of schemes, e.g., the prime factorization of RSA moduli or the discrete logarithm of an element in logarithm-based schemes. Furthermore, in the above examples the validity of the secret key can be verified in polynomial time. Yet, in other cases, e.g., commitment schemes, additional information aux may be required to verify the validity of secrets with respect to publicly available information. Typically, the randomness used in the key-generation process suffices. Moreover, in such schemes the key generation, e.g., the commit stage may be interactive. We then denote by pk the entire information that is published. This motivates our interest in schemes where a 3-ary relation $Pred$ with the following properties exists:

- *NP-recognition of secret keys*: $Pred$ is recognizable in polynomial-time. This implies that the languages
$$Pred_1 = \{pk | \exists sk, aux : (sk, aux, pk) \in Pred\}$$
of valid public keys and
$$Pred_2 = \{(sk, pk) | \exists aux : (sk, aux, pk) \in Pred\}$$
are in NP and therefore have zero-knowledge proof systems.

- *Unique secret keys*: For every pk, there exists at most one sk such that $(sk, pk) \in Pred_2$. If key generation is interactive, there may be an exponentially small error probability.

- Key generation of the given cryptographic scheme produces triples $(sk, aux, pk) \in Pred$.

The following lemma shows that the requirement of unique secret keys is no serious restriction.

Lemma 1 *Any cryptographic scheme where only computational secrecy of sk is required can be transformed into one with unique secret keys.*

Proof sketch: We use the commitment scheme from [17]. It is unconditionally binding, has a non-interactive opening stage, and can be based on any one-way function. In the commit stage, a random number is needed which may be chosen by a trusted server once and for all (see Section 5 in [17]). We modify the given cryptographic scheme by adding a commitment C on sk to the original public key pk and adding to aux the string *open* needed to open the commitment. The other algorithms of the scheme are not changed.

Uniqueness is clear by the binding property of the commitment, and the additional commitment does not harm the computational secrecy of sk. □

Construction 1 (generic, gradual, unbounded limitations indices): *Given a cryptographic scheme with unique secret keys with a relation $Pred$ as above.*

- *Server setup: For simplicity, we let the server choose a parameter n for the length of the secret keys once and for all. The server randomly generates a reference string $Ref \in \{0, 1\}^{\mathrm{poly}(n)}$ for the non-interactive zero-knowledge proof system and writes it in the public file. It also sets up a scheme that defines a sequence $r_1, r_2, ...$ of independently and uniformly chosen n-bit strings so that anyone can obtain r_l in $\mathrm{poly}(n, \log l)$ time. If the set of limitations indices is small, the sequence can simply be listed in the public record; other possibilities are mentioned below.*

- *Registration: Suppose a user has generated a primary key pair (sk, pk), together with the value aux so that $(sk, aux, pk) \in Pred$. The user proves to the server in zero-knowledge that pk is valid, i.e., lies in $Pred_1$. Now the user's public record is simply pk. In the case of interactive key generation, it has to be carried out during registration together with the server.*

- *Secondary key validation: The validation tag for this user's secondary key for the limitations index l is the pair (σ_l, π_l) where*

  1. *σ_l is the inner product modulo 2 of sk and r_l;*
  2. *π_l is a non-interactive zero-knowledge proof with respect to the reference string Ref that there exists sk so that $(sk, pk) \in Pred_2$ and σ_l equals the inner product modulo 2 of sk and r_l;*

  Recall that the authorization of the secondary key by the user himself, e.g., by signing (l, pk_l) using sk, is done outside this core part of our scheme.

- *Verification: The user gives the verifier an (interactive) zero-knowledge proof of knowledge [15, 1] that he knows (σ_l, π_l) such that π_l is a valid non-interactive zero-knowledge proof with respect to Ref and pk (which the verifier also knows) and the secret σ_l.*

An issue left open is how the r_l's become available when the set of limitations indices is large. In this case, we need a random function mapping limitations indices to n-bit strings. One possibility is to have the server maintain a "trusted beacon" which implements such a function. If we only consider polynomial-time adversaries, the server can use a pseudo-random function [12] instead of choosing and storing real random values. In practice, one may use a concrete publicly available function, typically a hash function, which is believed to "behave randomly" (cf. [10, 2]). Actually, we merely need a specific "random behaviour" as postulated in Assumption 1 below.

To quantify what is leaked by a single secondary key triple, we refer to the formalism of knowledge complexity [15, 14]. More precisely, we only quantify the knowledge leaked by the validation tag. (The secondary key pair as such is also knowledge, but this is not what we are concerned about because this key pair is generated independently of the self-delegation scheme.) Loosely speaking, one says that κ bits of knowledge are leaked by a protocol or a message with respect to a given state if whatever can be efficiently computed after the execution of the protocol or receipt of the message can also be computed efficiently with probability at least $2^{-\kappa}$ from the state itself, i.e., without executing the protocol or receiving the message. Thus, for small κ, more precisely $\kappa = O(\log n)$, if one can find the primary secret key with non-negligible probability after receiving at most κ bits of knowledge, one can do the same without receiving these bits: Non-negligible means that the probability is at least $1/\mathrm{poly}(n)$, and the additional factor $2^{-\kappa}$ is also at least $1/\mathrm{poly}(n)$; hence the overall success probability is still non-negligible.

Proposition 1 (generic, gradual, unbounded limitations indices):

1. Restricted damage from key disclosure: *The scheme leaks no knowledge except one bit in each correct validation tag. The user only needs to trust the server for the randomness of the reference string Ref.* (The above holds no matter which r_l's are used.)
2. Controlled propagation: *Suppose the user can access at most a certain number $m < 2^{n/2}$ of the r_l's. Then, with probability at least $1 - 2^{-n}$ (over the choice of the r_l's), if he gives away $2n$ valid secondary key triples to a second person, the second person can efficiently list a set of at most m^2 possibilities for the primary secret key.*

Observe that m can in practice be bounded in three ways: by the overall size of the set of limitations indices, the speed with which a beacon answers, and the running-time of the user. In particular, if the set of limitations indices is small, this means particularly strong propagation deterring.

The list of possible secret keys can be used in different ways: Firstly, the user can efficiently perform any task requiring this key with probability $1/m^2$, e.g., by uniformly selecting an element of this list. Secondly, whenever a guess at the secret key can be verified efficiently in the underlying scheme, which is the case if $Pred_2$ is efficiently recognizable or if auxiliary information can be used to recognize the secret key (i.e., the success of some activity that uses it), he can reconstruct the user's primary secret key in time $\mathrm{poly}(n) \cdot m^2$. An alternative for the remaining cases is to add another commitment on aux, the NP-witness for $(sk, pk) \in Pred_2$, and let the σ_l's extend over the pair (sk, aux).

For the proof of this and some other propositions, we need a technical lemma about the probability that too many of the r_l's are linearly dependent and therefore the resulting σ_l's do not allow reconstruction of the primary secret key. Throughout the text, arithmetic is in a finite field, even if elements are only selected from a subset of this field. Thus, when we talk of the rank of a matrix over a subset, we mean its rank over the field, i.e., the maximum number of linearly independent rows with respect to coefficients from the field.

Lemma 2 *Let m, n, t, R be integers so that $m > n > R$ and $m \geq t \geq R$, and q be a prime power. Let $S \subseteq \mathrm{GF}(q)$. Then, the probability $P_S(m, n, t, R)$ that a uniformly chosen m-by-n matrix over S contains t rows which together have rank at most R is bounded above by*

$$P_S(m, n, t, R) \leq \frac{(2m)^t}{t!} \cdot |S|^{-(n-R)\cdot(t-R)}.$$

The proof is given in the appendix.

Proof sketch of Proposition 1: We first prove Part 1, "restricted damage". First, we consider both the *registration* and *verification* protocols and show that the user is not giving away anything beyond the standard operation of the public-key infrastructure. Specifically, in the registration protocol, the user provides a zero-knowledge proof that its public key is valid, i.e., lies in $Pred_1$. Thus, this protocol leaks no additional knowledge. As for the verification protocol, the only thing which goes on here is a zero-knowledge proof (of knowledge).

Next, we consider what is given away when an adversary obtains a correctly validated secondary key triple. That is, we need to consider what is revealed on top of the secondary secret key itself. We claim that giving away such a validation tag, denoted (σ_l, π_l), merely yields one bit of knowledge. The claim holds since σ_l is merely a bit and π_l is merely a zero-knowledge proof referring to σ_l and the primary public key *pk*. (Recall that knowledge complexity is always bounded by the actual length of objects and that it is at most additive[14].)

Note that the user need not trust the server except with the random selection of Ref: What we showed so far does not rely on the randomness of the r_l's.

To prove the second part, "controlled propagation", we wish to show that it is infeasible for the user to find $2n$ limitation indices so that the corresponding r_l's, considered as vectors over GF(2), span a linear subspace of "small" dimension. Specifically, we consider a user who inspects at most m r_l's in an attempt to find $2n$ r_l's which span a subspace of dimension smaller than $n - 2\log_2 m$. Invoking Lemma 2, with $q = 2$, $S = \{0, 1\}$, $t = 2n$, and $R = n - 2\log_2 m$, we obtain that the probability that the m inspected r_l's, contain a subset of $2n$ vectors which span a linear subspace of dimension smaller than $n - 2\log_2 m$ is bounded by

$$P_{\{0,1\}}(m, n, 2n, R) \leq \frac{(2m)^{2n}}{(2n)!} \cdot 2^{-(2\log_2 m)\cdot(n+2\log_2 m)} < 2^{-n}.$$

From this point on, we ignore this negligible probability event. Without loss of generality, we can assume that the user will only give away secondary key triples for limitations indices l for which he has inspected r_l (because inspecting $2n$ additional values is no significant overhead). Thus, we may assume that the values r_l's corresponding to the $2n$ valid secondary key triples given to the second person span a linear subspace of dimension at least $n - 2\log_2 m$.

By the soundness of the interactive zero-knowledge proof of knowledge, having a valid secondary key triple, i.e., one that enables the second person to convince an honest verifier, implies that the second person knows a correct pair (σ_l, π_l). More formally, there is a universal extractor that effectively extracts this pair. Now the soundness of the (non-interactive zero-knowledge) proof π_l implies that σ_l is in fact the inner product of the *one* fixed *sk* and r_l. The uniqueness property of $Pred_2$ is used here to infer that all σ_l's refer to the same *sk*. Thus, the second person has a system of at least $n - 2\log_2 m$ independent linear equations modulo 2 about the n unknown bits of *sk*. He can easily list all the (at most m^2) solutions of the system, which correspond to up to m^2 different possible values of *sk*. ■

We note that a statement analogous to Proposition 1 holds also if the r_l's are determined by a hash function satisfying the following assumption, rather than being selected at random.

Assumption 1 (a good hash function): *Let $\mathcal{F} = \{\mathcal{F}_n\}_{n\in\mathbb{N}}$, with $\mathcal{F}_n$ consisting of functions which map binary strings into $\{0,1\}^n$, such that given* the description of *a function f in $\mathcal{F}_n$ it is infeasible to find indices $l_1, ..., l_{2n}$ such that the matrix defined by the strings $f(l_1), ..., f(l_{2n})$ has rank smaller than $n - O(\log n)$.*

Our next construction is only for a bounded set of limitations indices. Its advantages are that there is no gap between the two properties and that no restrictions in the user's access to an oracle are needed: If the primary secret key has $n = b \cdot \ell$ bits, we guarantee that the validation tag of each stolen secondary key gives away at most ℓ bits of knowledge, and on the other hand we guarantee that b secondary keys give all $n = b \cdot \ell$ bits of information (whereas above, one needed twice as many secondary keys, and obtained a small list containing the primary key rather than the key itself).

More importantly, the next construction does not require random strings r_l and so the issues involved with their generation vanish. In particular, there is no need to use a random function (in case of a huge set of limitations indices) or store many random values in the public file (in case of a small such set).

Additionally, the slope of the gradual degradation, i.e., the number ℓ, can be chosen somewhat freely between 1 and n (but we could also adapt Construction 1 to have that property). However, here this number also determines that the number of limitations indices is at most $2^\ell - 1$. Thus, this scheme is most adequate for cases where the set of limitations indices is rather small, since typically the user would like the gradual degradation slope to be small. For example, setting $\ell = 10$ allows 1023 limitation indices.

Construction 2 (generic, gradual, severely bounded limitations indices): *Given a cryptographic scheme with unique secret keys with a relation Pred as above.*

- *Server setup: For simplicity, let parameters n for the length of secret keys and ℓ for the slope of the degradation be chosen once and for all. Fix an easily computable enumeration $\alpha_1, ..., \alpha_{2^\ell-1}$ of the non-zero elements of $\mathrm{GF}(2^\ell)$. (Recall that this scheme only supports $2^\ell - 1$ possible limitations indices.) Choose a reference string $Ref \in \{0,1\}^{\mathrm{poly}(n)}$ for the non-interactive zero-knowledge proof system.*

- *Registration: Again, the public record is simply the user's public key pk, whose validity, i.e., lying in $Pred_1$, the user must prove to the server in zero-knowledge.*

- *Secondary key validation: The user's primary secret key is represented as a b-long sequence of ℓ-bit blocks $sk = (s_0, ..., s_{b-1})$. The validation tag for this user's secondary key for the limitations index l is the pair (σ_l, π_l) where*

  1. *σ_l is the value of the polynomial $p(x) \stackrel{\text{def}}{=} \sum_{j=0}^{b-1} s_j x^j$, over $\mathrm{GF}(2^\ell)$, at the point α_l;*
  2. *π_l is a non-interactive zero-knowledge proof with respect to the reference string Ref that there exists sk so that $(sk, pk) \in Pred_2$ and $\sigma_l = \sum_{j=0}^{b-1} s_j \alpha_l^j$.*

- *Verification: As in Construction 1, the user gives a zero-knowledge proof of knowledge of a correct pair (σ_l, π_l).*

Proposition 2 (generic, gradual, severely bounded limitations indices):

1. Restricted damage from key disclosure: *The scheme leaks no knowledge except at most ℓ bits in each correct validation tag. The user need not trust the server except for the randomness of the reference string Ref.*

2. Controlled propagation: *Any b valid secondary key triples yield the primary secret key.*

Proof sketch: Part 1 is proved just as in Proposition 1, this time with σ_l of length ℓ.

To prove the second part, we note that for any b non-zero α_l's, the system of equations given in Item 1 of the validation protocol, considered as linear equations with the unknowns s_j, is a Vandermonde matrix and thus always non-singular.

As in the proof of Proposition 1, the soundness of the two proof systems implies that a second person who has valid secondary key triples knows correct values σ_l. Thus, the second person has a system of b independent linear equations in the b (ℓ-bit) blocks of the primary secret key, which he can easily and uniquely solve. □

In the next construction, the set of limitations indices is also bounded, but this bound may be larger than before: It is now 2^n, where n is the length of the keys, rather than 2^ℓ, where $\ell \leq n$ (and typically $\ell \ll n$) was the number of bits leaked per secondary secret key. This construction has a threshold k that can be chosen very freely, i.e., between 1 and $2^n - 1$.

Construction 3 (generic, threshold, bounded): *Given a cryptographic scheme with unique secret keys with a relation $Pred$ as above. Additionally, we need a secure commitment scheme that is unconditionally binding.*

- *Server setup: For simplicity, let parameters n for the length of secret keys and k for the threshold be chosen once and for all. Fix an easily computable enumeration $\alpha_1, ..., \alpha_{2^n-1}$ of the non-zero elements of $\mathrm{GF}(2^n)$. (Recall that this scheme only supports $2^n - 1$ possible limitations indices.) Choose a reference string $Ref \in \{0,1\}^{\mathrm{poly}(n)}$ for the non-interactive zero-knowledge proof system.*

- *Registration: Suppose a user has generated a primary key pair (sk, pk), together with the value aux so that $(sk, aux, pk) \in Pred$. It first proves to the server in zero-knowledge that pk is valid, i.e., lies in $Pred_1$.*

  Next, it uniformly selects a polynomial of degree $k-1$ over $\mathrm{GF}(2^n)$ with the primary secret key as the constant coefficient, say, $p(x) \stackrel{\text{def}}{=} \sum_{j=0}^{k-1} p_j \cdot x^j$ with $p_0 = sk$. This corresponds to setting up a secret sharing scheme as in [20]. He makes a commitment C to all the non-constant coefficients using the given commitment scheme. Typically, the length of the commitment automatically shows that the content is a polynomial of degree $k-1$, otherwise this must be shown in zero-knowledge.

  The user's public record is (pk, C).

- *Secondary key validation: The validation tag for this user's secondary key for the limitations index l is the pair (σ_l, π_l) where*

  1. *σ_l is the value of the polynomial p at the point α_l;*

2. *π_l is a non-interactive zero-knowledge proof with respect to the reference string Ref that there exist sk and p so that $(sk, pk) \in Pred_2$, the non-constant coefficients of p correspond to the commitment string C, the constant term ($p_0 = p(0)$) equals sk, and $\sigma_l = p(\alpha_l)$.*

- *Verification: The user gives a zero-knowledge proof that he knows a correct pair (σ_l, π_l).*

Proposition 3 (generic, threshold, bounded limitations indices):

1. Restricted damage from key disclosure: *Public information and the validation tags of at most $k-1$ correctly validated secondary key triples leak no knowledge. The user need not trust the server except for the randomness of the reference string Ref.*

2. Controlled propagation: *Any k valid secondary key triples yield the primary secret key.*

Proof sketch: The second part follows as in Proposition 2. For Part 1, the security of the commitment scheme and the zero-knowledge proofs implies that the only non-negligible source of knowledge is in the values σ_i. These are $k-1$ shares of a secret sharing scheme with threshold k and therefore give no information about the secret pk [20]. □

4 Discrete-Logarithm Schemes

We now present two versions of a practical self-delegation scheme for cryptographic schemes based on discrete logarithms in groups of prime order. Important examples are Schnorr and DSS signatures [19, 6]. The advantage over the schemes in the previous section is that no general zero-knowledge proof techniques are needed, and the resulting self-delegation schemes are not much less efficient than the underlying schemes.

More precisely, we assume an underlying cryptographic scheme with the following key generation: First a prime q is chosen, typically 160 bits long; next a prime p is chosen, typically 512 bits, such that q divides $p-1$; and then an element g of order q is chosen in the multiplicative group modulo p. These choices may be made once and for all by the server (with suitable precautions), or by each user individually. A secret key is a uniformly chosen value $sk \in \mathrm{GF}(q)^* = \{1, ..., q\}$, and the corresponding main part of the public key is $h = g^{sk} \bmod p$.

The basic idea in the following schemes is somewhat similar to the Verifiable Secret Sharing scheme based on discrete logarithms [8, 18]. Each valid secondary key triple will give one linear combination of the shares, such that enough secondary keys will yield an equation system that can be uniquely solved for the shares and thus for the primary secret key. The schemes differ in the way the coefficients of the linear equations are chosen.

Both schemes have a threshold k so that the validation tags of less than k stolen secondary keys do not give any knowledge. In the second construction, this is a threshold in the strong sense, i.e., any larger number of valid secondary key triples yields the primary public key, whereas the first construction has a second threshold $2k$ from where on this is guaranteed. We stress that any $k \geq 2$ is possible. The advantages of the first version are having an unbounded set of limitations indices, and being flexible with respect to the choice of coefficients. In particular, the coefficients may be chosen from a set of rather small numbers (modulo q), so that exponentiating with these coefficients is relatively cheap.

There are two ways of using both these schemes: The results of the core scheme can either be used as validation tags for independently generated secondary key pairs, just as in our generic constructions, or we can use them directly as secondary keys. We mainly describe the first variant. Here, the secondary keys can even be from a different cryptographic scheme; only the primary key pair has to correspond to the discrete-logarithm-based key generation described above.

Construction 4 (discrete log, unbounded limitations indices): *Let any cryptographic scheme based on discrete logarithms in groups of prime order be given, i.e., essentially the key generation as described above.*

- *Server setup: For simplicity, we let the server generate the parameters (q, p, g) of the underlying scheme. It also selects a set $S \subset \mathrm{GF}(q)$; the appropriate size in relation to the other parameters will be discussed below. It sets up a scheme that defines a sequence $r_1, r_2, \ldots$ of independently and uniformly chosen k-tuples over S so that anyone can obtain r_l in $\mathrm{poly}(k, \log q, \log l)$ time. Ways of doing this were described in and after Construction 1. We write an individual tuple as $r_l = (r_{l,1}, \ldots, r_{l,k})$.*

- *Registration: Given a primary key pair (sk, pk) from the underlying discrete logarithm scheme, the user randomly selects a tuple $(s_1, \ldots, s_k)$ over $\mathrm{GF}(q)$ with the constraint $sk = \sum_{j=1}^{k} s_j \bmod q$. This tuple is his extended primary secret key. His public record consists of the values $h_j = g^{s_j} \bmod p$, for $j = 1, \ldots, k$. The server verifies that $pk = \prod_{j=1}^{k} h_j \bmod p$.*

- *Secondary key validation: The validation tag for this user's secondary key for the limitations index l is the value $val_l = sk_l^*$ where*

  $$sk_l^* = \sum_{j=1}^{k} r_{l,j} s_j \bmod q.$$

  Note that this validation tag can be seen as a secret key of the underlying discrete logarithm scheme; the corresponding public key would be $pk_l^* = g^{sk_l^*} \bmod p$.

- *Verification: The verifier can compute this value pk_l^* as*

  $$pk_l^* = \prod_{j=1}^{k} h_j^{r_{l,j}} \bmod p$$

  using the values h_j in the user's public record. Now the user has to give a proof of knowledge of sk_l^*, i.e., of the discrete logarithm of pk_l^*. The proof better be in zero-knowledge. The system from [5] is quite efficient, but it needs t rounds of communication (each with one exponentiation on both sides) for a security level of 2^{-t}. Actually, some of these can be executed in parallel with the usual trade-off between amount of parallelism and tightness of the security, cf. [10]. However, t need not be very large, because we do not try to exclude every single case of propagation anyway. An alternative is Schnorr identification [19], which needs only one exponentiation on each side, but is only known to be witness-hiding [9].

As stated above, a way of using this core scheme is to use the values sk_l^* directly as the secondary secret keys. This improves efficiency in case we can omit the proof of knowledge (as the subsequent usage of sk_l^* establishes knowledge of it). However, in such a case we have to be careful how the security of the underlying scheme and our self-delegation scheme influence each other. In particular, the legitimate use of sk_l^* in the underlying scheme is typically not zero-knowledge and might therefore *additionally* endanger the primary secret key sk via our self-delegation scheme.

Thus security has to be considered specifically for each concrete underlying scheme, e.g., for Schnorr or DSS signatures. Specifically, for Schnorr identification security seems to be hold. The following proposition refers to the security of the core self-delegation scheme itself.

Proposition 4 (discrete log, unbounded limitations indices):

1. Restricted damage from key disclosure: *For any* given *set of at most $k' < k$ limitations indices, with probability $1 - |S|^{-(k-k')}$ (over the choice of the r_l's), the scheme leaks no knowledge to a thief who obtains the correct validation tags for these limitations indices, and it is infeasible for him to produce a valid secondary key triple for any given other limitations index l^* if computing discrete logarithms is hard.*

   The user has to trust the server for the randomness of the values r_l here.

2. Controlled propagation: *Suppose the user can access at most $m < \sqrt{|S|}$ of the r_l's. Then, with probability at least $1 - 2^{-k} \cdot |S|^{-1}$, if he gives a second person $2k$ valid secondary key triples, then the second person can easily derive the user's primary secret key.*

If the set of limitations indices is really unbounded and the bound on m is only given by the running-time of the user, we will simply use the set $S = \mathrm{GF}(q)$. Then the bound on m is realistic and the error probabilities in both properties are at most q^{-1}. If there is a fairly small set of limitations indices, denoted L, we can choose $S = 0, ..., |L|^2$ to have small exponents. Then, for reasonably small error probabilities, the threshold in the sense of the model will be what was called k' above, i.e., k will be chosen as $k' + k''$ such that $|L|^{-2k''}$ and $2^{-k}|L|^{-2}$ are considered small enough.

Proof sketch: For the proof of the first part, we first consider the probability that a specific vector, either r_{l^*} or $(1, ..., 1)$ (which plays a similar role by being the coefficients for sk), resides in the linear space spanned by the at most $k' < k$ vectors r_l for the given limitations indices. This probability is easily upper bounded by $|S|^{-(k-k')}$ (as in the proof of Lemma 2). From now on, we assume that the other, more likely case occurs.

The core part of the proof is now that the entire view a thief gets is perfect zero-knowledge. More precisely, we consider the following protocol between a user and a thief (we have already omitted the zero-knowledge proof in verification):

- The common inputs are pk and the given r_l's, and the user has the auxiliary input sk.
- The user randomly shares sk as $(s_1, ..., s_k)$ as above.
- He publishes the public record and "sends" the k' validation tags for the given limitations indices.

We have to construct a simulator for this protocol. Let M be the matrix consisting of the k' rows r_l and one additional row $(1, ..., 1)$. Without loss of generality, we can now assume that $k' = k - 1$ and M is of rank k (otherwise we can give the thief additional information to bring him into this situation). Anyone can easily compute the inverse M^{-1}. For each given vector of the $k - 1$ tags and the secret key, one can easily compute the only possible corresponding tuple $(s_1, ..., s_k)$ by multiplying the given vector with M^{-1}. The simulator works as follows:

1. It randomly selects tags sk_l^* for the $k - 1$ given limitations indices. For simplicity, we denote these limitations indices by $1, ..., k - 1$.

2. It computes $pk_l^* = g^{sk_l^*}$ for $l = 1, ..., k-1$.

3. Not knowing sk, it applies M^{-1} to the exponentiated values instead of the discrete logarithms. This yields equations of the following form

$$h_j = g^{s_j} = g^{(\sum_{l<k} m_{j,l}^* sk_l^*) + m_{j,k}^* sk} = \left(\prod_{l<k} (pk_l^*)^{m_{j,l}^*}\right) \cdot pk^{m_{j,k}^*},$$

where the $m_{j,l}^*$'s are the elements of M^{-1}.

4. It publishes these values h_j and the originally chosen tags.

It is easy to see that the results of this simulator are distributed exactly like those of the original protocol. This ensures that the self-delegation scheme does not endanger the security of the underlying scheme.

By a similar simulation argument, if a thief could compute a correct validation tag for a linearly independent r_{l^*}, he could also compute it from $pk_{l^*}^*$ alone, i.e., compute discrete logarithms. The proof of knowledge in verification guarantees that having a valid secondary key triple implies knowledge of such a correct validation tag.

To prove the second part, we first invoke Lemma 2 with the given set S, $n = k$, $t = 2k$ and $R = k-1$: The probability that the m inspected r_l's contain a subset of $2k$ vectors which span a linear subspace of dimension at most $k-1$ is bounded by

$$P_S(m, k, 2k, k-1) \leq \frac{(2m)^{2k}}{(2k)!} \cdot |S|^{-(k+1)} \leq \frac{2^{2k}}{(2k)!} \cdot |S|^{-1} < 2^{-k} \cdot |S|^{-1}.$$

Otherwise, we may assume that the r_l's used in the validation of the $2k$ secondary keys span a linear subspace of dimension k. The second person therefore has a system of k independent linear equations over GF(q) in the k unknown values s_j, which he can easily solve. □

The next construction has stronger security: There is no gap between the thresholds in the two properties and no additional error probabilities. Moreover, similar to Constructions 2 and 3, there are none of the practical disadvantages of needing trusted random numbers. The price is that the set of limitations indices is bounded. However, the bound is large, q, and should be sufficient for most practical purposes because one can hash to length $|q|$ whatever one really wanted to express in the limitations.

Construction 5 (discrete log, bounded limitations indices): *The construction is identical to Construction 4, except that the server does not set up any random values. Instead, fixed tuples of successive powers are used. For similarity of notation, we call them r_l, too. Concretely, we set $r_{l,j} = l^{j-1} \bmod q$, while excluding $l = 1$* (because the equation for that case defines the primary secret key).

Proposition 5 (discrete log, bounded limitations indices):

1. Restricted damage from key disclosure: *The scheme leaks no knowledge to a thief who obtains less than k correctly validated secondary key triples, and it is infeasible for him to produce a valid secondary key triple for any other limitations index l^* if computing discrete logarithms is hard. The user need not trust the server at all.*

2. Controlled propagation: *Any k valid secondary key triples yield the primary secret key.*

Proof sketch: The main observation is that the system of equations defined by the r_l's is a Vandermonde matrix and thus of rank k'', where $k'' \leq k$ is the number of given secondary secret keys. Then both parts follow as in the proof of Proposition 4. $\square$

The efficiency of our discrete-logarithm self-delegation schemes can be summarized as follows: The use of a primary or secondary key pair, once it has been registered or verified, respectively, is exactly as in the underlying scheme. Registration (executed very rarely) requires k exponentiations by the user, where k is the threshold. Secondary key validation is essentially as efficient as in the underlying scheme. In verification of a secondary key, the basic part does not involve the user and needs k exponentiations by the verifier. In addition, a zero-knowledge proof of knowledge of a discrete logarithm is needed, but it will often be replaceable by Schnorr identification in practice.

5 Conclusion

We have formulated a new and very realistic cryptographic problem – self-delegation with controlled propagation – and have presented several schemes for solving this problem. Our solutions discourage the user from delegating rights that have been given to him (for personal use only) to others, because such delegation "leaks" information about his primary secret key. Effective control of the leakage rate provides a way to balance between the damage to the user caused by losing his secondary keys, and the risk to the information supplier or other access-controlled system of the user delegating away such access rights.

References

[1] M. Bellare and O. Goldreich: On Defining Proofs of Knowledge; in *Crypto '92*, Springer-Verlag, LNCS Vol. 740, pp. 390-420, 1992.

[2] M. Bellare and P. Rogaway: Random oracles are practical: a paradigm for designing efficient protocols; in Proceedings of *1st Annual Conference on Computer and Communications Security*, ACM, pp. 62-73, 1993.

[3] M. Blum: How to Exchange (Secret) Keys; *ACM Transactions on Computer Systems*, Vol. 1, No. 2, pp. 175-193, 1983.

[4] M. Blum, P. Feldman, and S. Micali: Non-Interactive Zero-Knowledge and its Applications; in *20th STOC*, pp. 103-112, 1988.

[5] D. Chaum, J.-H. Evertse, and J. van de Graaf: An improved protocol for demonstrating possession of discrete logarithms and some generalizations; in *Eurocrypt '87*, Springer-Verlag, LNCS Vol. 304, pp. 127-141, 1988.

[6] The Digital Signature Standard Proposed by NIST; *Communications of the ACM*, Vol. 35, No. 7, pp. 36-40, 1992.

[7] U. Feige, D. Lapidot, and A. Shamir: Multiple non-interactive zero knowledge proofs based on a single random string; in *31st FOCS*, pp. 308-317, 1990.

[8] P. Feldman: A practical scheme for non-interactive verifiable secret sharing; in *20th FOCS*, pp. 427-437, 1987.

[9] U. Feige and A. Shamir: Witness Indistinguishability and Witness Hiding Protocols; inn *22nd STOC*, pp. 416-426, 1990.

[10] A. Fiat and A. Shamir: How to Prove Yourself: Practical Solutions to Identification and Signature Problems; in *Crypto '86*, Springer-Verlag, LNCS Vol. 263, pp. 186-194, 1987.

[11] M. Gasser, A. Goldstein, Ch. Kaufman, and B. Lampson: The Digital Distributed System Security Architecture; in Proceedings of *12th National Computer Security Conference*, pp. 305-319, 1989.

[12] O. Goldreich, S. Goldwasser, S. Micali: How to Construct Random Functions; *Journal of the ACM*, Vol. 33, No. 4, pp. 792-807, 1986. Extended abstract in *25th FOCS*, 1984.

[13] O. Goldreich, S. Micali, and A. Wigderson: Proofs that Yield Nothing but their Validity or All Languages in NP Have Zero-Knowledge Proof Systems; *Journal of the ACM*, Vol. 38, No. 1, pp. 691-729, 1991. Extended abstract in *27th FOCS*, 1986.

[14] O. Goldreich and E. Petrank: Quantifying Knowledge Complexity; in *32nd FOCS*, pp. 59-68, 1991. To appear in *Computational Complexity.*

[15] S. Goldwasser, S. Micali, and C. Rackoff: The Knowledge Complexity of Interactive Proof Systems; *SIAM Journal on Computing*, Vol. 18, No. 1, pp. 186-208, 1989. Extended abstract in *17th STOC*, 1985.

[16] J. Kilian and E. Petrank: An Efficient Non-Interactive Zero-Knowledge Proof System for NP with General Assumptions; to appear in *Journal of Cryptography.* Available as TR95-038 of *ECCC (Electronic Colloquium on Computational Complexity)*, `http://www.eccc.uni-trier.de/eccc/`, 1995.

[17] M. Naor: Bit Commitment Using Pseudorandomness; *Journal of Cryptology*, Vol. 4, No. 2, pp. 151-158, 1991.

[18] T. P. Pedersen: Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing; in *Crypto '91*, Springer-Verlag, LNCS Vol. 576, pp. 129-140, 1992.

[19] C. P. Schnorr: Efficient Signature Generation by Smart Cards; *Journal of Cryptology*, Vol. 4, No. 3, pp. 161-174, 1991.

[20] A. Shamir: How to Share a Secret; *Communications of the ACM*, Vol. 22, No. 11, pp. 612-613, 1979.

Appendix: Proof of Lemma 1

The following lemma has to be proved:

Lemma 2 *Let m, n, t, R be integers so that $m > n > R$ and $m \geq t \geq R$, and q be a prime power. Let $S \subseteq \mathrm{GF}(q)$. Then, the probability $P_S(m,n,t,R)$ that a uniformly chosen m-by-n matrix over S contains t rows which together have rank at most R is bounded above by*

$$P_S(m,n,t,R) \leq \frac{(2m)^t}{t!} \cdot |S|^{-(n-R)\cdot(t-R)}.$$

Recall that arithmetic is always in a (prime) finite field, even though elements are selected from the subset S of the field.

Proof: Let $P_S(n,t,r)$ denote the probability that a uniformly chosen t-by-n matrix over S has rank r. Then, our desired probability is upper bounded by

$$P_S(m,n,t,R) \;\leq\; \binom{m}{t} \cdot \sum_{r=0}^{R} P_S(n,t,r) \;<\; \frac{m^t}{t!} \cdot \sum_{r=0}^{R} P_S(n,t,r). \tag{1}$$

Next we bound $P_S(n,t,r)$ using the probability that $t-r$ rows (over S) are in the linear space spanned by r other rows. For any r-by-n matrix M, let P_S^M denote the probability that a uniformly selected n-dimensional vector over $S \subseteq \mathrm{GF}(q)$ resides in the linear subspace $\langle M \rangle$ spanned by M. We have

$$\begin{aligned} P_S(n,t,r) \;&\leq\; \binom{t}{r} \cdot \sum_{M \in S^{r\times n}:\mathrm{rk}(M)=r} \mathrm{Prob}(\text{matrix } M \text{ is chosen}) \cdot (P_S^M)^{t-r} \\ &\leq\; \binom{t}{r} \cdot \max_{M \in S^{r\times n}:\mathrm{rk}(M)=r} \left\{ (P_S^M)^{t-r} \right\}. \end{aligned}$$

To bound P_S^M for a matrix M of rank r, we consider a set of $n-r$ row-reduced equations whose solution space is precisely $\langle M \rangle$. (Readers unfamiliar with dual vectorspaces may think of M as the generator matrix of a linear code and the equations as a corresponding checkmatrix. Row-reduced means that there is an identity submatrix, but not necessarily in adjacent columns.) The probability that a uniformly chosen vector over $S \subseteq \mathrm{GF}(q)$ fulfills these equations is at most $1/|S|^{n-r}$: First randomly set the r elements not corresponding to the identity submatrix. The remaining ones are then uniquely determined, and the solution may not even lie within S^n. We therefore get

$$P_S(n,t,r) \;\leq\; \binom{t}{r} \cdot |S|^{-(n-r)\cdot(t-r)} \;\leq\; \binom{t}{r} \cdot |S|^{-(n-R)\cdot(t-R)}.$$

Substituting this into Eq. (1), we obtain

$$P_S(m,n,t,R) \;\leq\; \frac{m^t}{t!} \cdot 2^t \cdot |S|^{-(n-R)\cdot(t-R)},$$

which proves the lemma. ■