Journal of Universal Computer Science, vol. 14, no. 5 (2008), 693-716
submitted: 1/11/07, accepted: 28/2/08, appeared: 1/3/08 © *J.UCS*

Enhancing ZRTP by using Computational Puzzles

Helmut Hlavacs, Wilfried Gansterer, Hannes Schabauer
Joachim Zottl
(University of Vienna, Austria
{helmut.hlavacs, wilfried.gansterer, hannes.schabauer,
joachim.zottl}@univie.ac.at)

Martin Petraschek, Thomas Hoeher, Oliver Jung
(ftw (Telecommunications Research Center Vienna), Austria
{petraschek, hoeher, jung}@ftw.at)

Abstract: In this paper we present and discuss a new approach for securing multimedia communication, which is based on three innovations. The first innovation is the integration of a challenge-response scheme for enhancing the Diffie-Hellman based ZRTP protocol. When being called, a callee must present the result of a computational puzzle (a "token") within a short amount of time. A Man-in-the-Middle (MitM) would not be able to compute such a token within the required time, and thus fail to get into the media path. The scheme works best in situations when ZRTP is most vulnerable to so-called Mafia Attacks, i.e., if both caller and callee do not know each other.

The second innovation complements the first one on those occasions where the above scheme may fail. The call is delayed for a certain amount of time which depends on the agreed session key. Since during a MitM attack two different keys (and thus waiting times) exist, caller and callee would not start their call at the same time and the MitM attack would fail.

The third innovation is in the definition of a new computational puzzle which forms the basis of the challenge-response scheme. We propose a computational puzzle which is based on computing selected eigenvectors of real symmetric matrices. In contrast to existing puzzles, the one we propose does not rely on a shared secret, can be validated quickly, and existing solution methods exhibit limited scalability so that the threat from attacks based on massively parallel computing resources can be controlled.

Key Words: VoIP, SRTP, ZRTP, computational puzzle, challenge-response, eigenvectors, call delay
Category: C.2.0, K.6.5

1 Introduction

In this paper we show how to drastically increase the security of Diffie-Hellman based approaches like ZRTP. We propose a combination of three innovations, the first, challenge-response based on computational puzzles, is completely new. The second called Random Call Delay has been known before in a different variant. The third innovation is a new form of computational puzzle that is difficult to be computed in parallel, and can be used for our new challenge-response scheme.

2 Diffie-Hellman and ZRTP

2.1 Diffie-Hellman key exchange protocol

In 1976, Whitfield Diffie, Martin E. Hellman and Ralph Merkel (rarely mentioned) published a cryptographic protocol scheme to agree upon a shared key over an insecure channel. In other words, their approach enables two communication partners to negotiate a symmetric secret even though anyone could wiretap this public procedure.

Figure 1 represents the typical two-stage procedure where Alice acts as initiator, providing Bob the shared public values n (residue class) and g (generator/base) as well as its individual public value X (step 1).

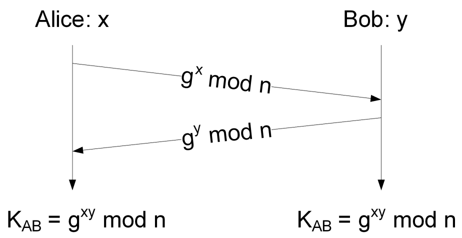

Figure 1: Diffie-Hellman key agreement

This is calculated by

$$X = g^x \bmod n,$$

where x is the private value which must be kept secret by Alice. Once Bob receives the shared public values he on his part calculates the individual public value

$$Y = g^y \bmod n$$

and sends it back to Alice, again keeping the value y secret to himself (step 2). Usually, the two public values (n, g) are known in advance as they have no additional security relevance. If so, then both parties can pre-calculate their public values X and Y, respectively and the Diffie-Hellman protocol only requires to exchange these values. By means of this agreement Alice and Bob have the corresponding information to calculate a common shared secret

$$K_{AB} = X^y \bmod n = Y^x \bmod n = g^{xy} \bmod n.$$

As the Diffie-Hellman key exchange protocol does not provide any measure to prove identity, it is susceptible to impersonation attacks like *Man-in-the-Middle* (aka Bucket-Brigade attack or Mafia Attack). From the practical viewpoint once an attacker is able to intercept the communication path between Alice and Bob an attack on the key agreement can be launched. In such a case the Man-in-the-Middle (MitM) acts as additional but invisible communication partner using his own secret value z, that negotiates two *different* shared keys, a key K_{AM} with Alice and a key K_{MB} with Bob. This means that all traffic between the victims is decrypted and newly encrypted by the MitM to have access to the original data (Figure 2).

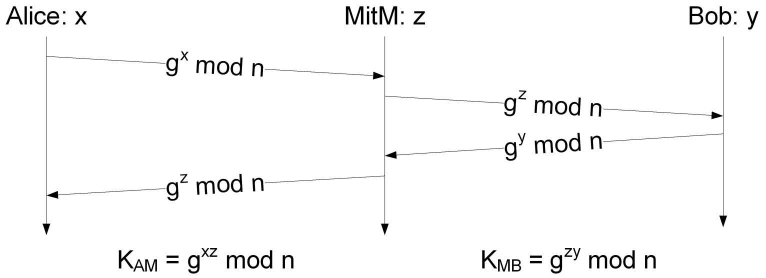

Figure 2: A MitM-Attack results in two different secret keys K_{AM} and K_{MB}.

In the following section the protocol ZRTP is dealt with that tries to eliminate this weakness by comparing a hashed image of the shared key called Short Authentication String (SAS) over the audio channel.

2.2 ZRTP

ZRTP is an innovative approach for VoIP media path encryption. In 2006, Phil Zimmermann submitted the draft "ZRTP: Extensions to RTP for Diffie-Hellman Key Agreement for SRTP" to the IETF even though the key idea originates from PGPfone already developed in 1995. However, at that time the Internet and in particular the VoIP technology was not widely used. The abbreviation ZRTP combines a 'Z' for Zimmermann with the established protocol for media transmission RTP (Real-time Transport Protocol). ZRTP is implemented in the free Zfone product. It is a kind of "bump-in-the-stack" application that passively observes all network traffic and once VoIP signaling is detected it starts active operation. ZRTP is then provided on the negotiated media ports. Due to this implementation Zfone is able to cooperate with each softphone as it works in a completely transparent way for upper layers.

ZRTP is a VoIP encryption protocol that relies on the Diffie-Hellman key agreement (see Section 2.1). The smartness of this approach is that the exchange of session keys is fully integrated into the media path (in-band keying within RTP). This solution follows the basic Internet concept of peer-to-peer computing as neither SIP signaling nor any servers are required. In the latest ZRTP draft Zimmermann also defined Signaling Interaction, however solely to provide an add-on in terms of discovery and authentication. The protocol works as follows:

1. Alice acting as initiator sends a Hello-message containing primarily her ZID (ZRTP ID) and several complementary parameters:

$$Hello\{version, options, Alice's\ ZID\}.$$

2. Bob confirms the reception by sending $HelloAck\{\}$, and subsequently sends a Hello-message by himself:

$$Hello\{version, options, Bob's\ ZID\}.$$

3. Alice also confirms the reception by sending $HelloAck\{\}$. Afterwards, the Diffie-Hellmann key agreement starts with a Commit-message

$$Commit\{Alice's\ ZID, options, hash(X)\}.$$

   Hereby, Alice commits to the public value X without actually sending it. This measure foils SAS (see below) collision attacks.

4. The hash commitment signals Bob to start the two-step DH key exchange with the DHPart1-message:

$$DHPart1\{Y(public\ value), (old)\ shared\ secret\ hashes\}.$$

   The out-dated hashed key material is used to implement the property of *key continuity*. Once Alice and Bob have established an authenticated shared key (see SAS comparison) all subsequent ZRTP sessions can be considered as secure.

5. Alice completes the Diffie-Hellmann procedure with the DHPart2-message:

$$DHPart2\{X(public\ value), (old)\ shared\ secret\ hashes\}.$$

   Now, the shared master key (s0) for SRTP (Secure RTP) is calculated by hashing the agreed DH-key, the old key hashes as well as some additional parameters.

6. From now on the communication channel between Alice and Bob is encrypted with SRTP which typically uses AES. In order to guarantee that the Diffie-Hellmann key exchange was not compromised by a MitM, ZRTP provides a mechanism named Short Authentication String (SAS). It is a compact

representation of the *exchanged master key* which is shown to both communication partners to be *read out aloud* over the telephone. Within the ZRTP Commit-message a so called SAS type is agreed upon which defines the actual SAS representation. Currently, either the leftmost 20 bits of the SAS value are z-base-32 encoded[1] which yields four characters, or the leftmost 16 bits are mapped to the PGP Wordlist.[2] If both strings match then it is very likely that there is no Man-in-the-Middle attack going on since the two different keys due to a MitM attack (Figure 2) would result in *two different* SAS values.

7. Finally, a three-way handshake consisting of Confirm1, Confirm2 and Confirm2Ack is applied to securely acknowledge the successful key agreement and to transmit some important encrypted parameters

$$Confirm1\{HMAC, D, S, Vflags, sig\}$$
$$Confirm2\{HMAC, D, S, Vflags, sig\}$$
$$Confirm2Ack\{\}.$$

Additionally, as each ZRTP session carries out a new Diffie-Hellman key agreement and at the end of each session the current keys are destroyed the property of *perfect forward secrecy* is provided, i.e., having access to any used session key does not enable an attacker to decrypt previous or following calls.

As a result, ZRTP is a smart protocol to allow encrypted VoIP communication based on SRTP without requiring the overhead of a Public Key Infrastructure (PKI), and with it the complexity of certificate and key management. The shared keys are agreed within the media channel in pure end-to-end manner using the Diffie-Hellman key exchange protocol with hash-commitment. For authentication purposes and MitM protection ZRTP introduces the comparison of an SAS value to be read aloud by both caller and callee.

However, as discussed in Section 2.1 Diffie-Hellman and also ZRTP are prone to Man-in-the-Middle attacks, and a MitM may interfere reading the SAS values, for example, by creating two separate calls and repeating everything heard from one caller to the other. Such an attack is likely to fail if both persons know each other's voice well, such as friends, relatives, colleagues etc. However, in case two strangers talk to each other, the MitM might succeed.

3 Challenge-Response using Computational Puzzles

Our proposed approach is an enhancement of Diffie-Hellman, and uses a new type of challenge-response scheme, i.e., the presentation of the result of a computational puzzle in time. As we will point out later, our approach does not guarantee

[1] http://zooko.com/repos/z-base-32/base32/DESIGN
[2] http://en.wikipedia.org/wiki/PGP_Words

100% authentication. However, it works best if both parties are strangers and have no known relationship, a situation that is perfect for an attack on ZRTP. When only taking into account such situations, and assuming that most of them are vulnerable to MitM attacks, then our approach will drastically reduce the percentage of successful attacks to a very small number.

Our challenge is communicated within the control channel, for instance SIP. Suppose Alice sends an INVITE(X) message to Bob, X being her public Diffie-Hellman key. Alice also requests Bob to send within a critical time dt_c, for instance 10 seconds, the result $R(P)$ of a computational puzzle P to Alice. Only if Bob is able to present the result of such a puzzle to Alice, Alice accepts Bob as further trustable, and the ZRTP media channel procedure can proceed. Otherwise, the second scheme Random Call Delay as described in Section 5 must be invoked. The puzzle result which we call *token* also must contain some additional information about Bob, for instance an additional public key that establishes a VPN between Alice and Bob, and the ZRTP data is sent through this additional encryption channel.

One important property of the computational puzzle is the fact that even the best supercomputer in the world is unable to compute the result within dt_c seconds. This requires that Bob has precomputed this result, for instance while harvesting unused cycles of his private PC during the night. The puzzle $P(T, U, Y)$ is defined through the following data:

- Period T of validity. Each token is valid only for a certain time, for instance only on Sept. 1st 2007.
- URI U of Bob.
- Temporary public key Y (belonging to a corresponding private key y) that is used for creating a VPN.

In order to be able to construct $P(T, U, Y)$, all input parameters including the time period T, URI U and public key Y must be known. If the MitM does not know Bob (if Alice and Bob are strangers this is quite likely) then the MitM cannot construct and precompute it.

Our approach relies on the assumption that if Alice and Bob do not know each other, the MitM is unlikely to be able to relate them, and predict that Alice in the near future might call Bob. In such a situation, if Alice calls Bob, and a MitM is sitting in between, the MitM would not be able to produce a token within the tolerance of dt_c seconds, even if commanding vast amounts of computing power. Thus, the MitM would not be able to create a token for a VPN between him and Alice. It is obvious that a MitM is not able to precompute tokens for a theoretically unlimited number of VoIP URIs for each day.

On the other hand, if Alice and Bob know each other (and thus each other's voice), then the MitM is likely to know this relation also, and is able to precompute one token for every day and each known acquaintance of Alice. This already requires commanding a powerful computing infrastructure. However, as stated above, in such a situation ZRTP is well suited to prevent a successful MitM attack.

3.1 Scenario 1: MitM shields Alice

The standard scenario (scenario 1) for our proposed scheme is shown in Figure 3. In scenario 1 we assume that an extremely powerful attacker is able to completely *shield* Alice, i.e., he is able to catch and manipulate all data packets sent to or from Alice, for instance in order to compromise the Diffie-Hellman key exchange. Alice sends the INVITE(X) to Bob, and within dt_c seconds, Bob must answer by

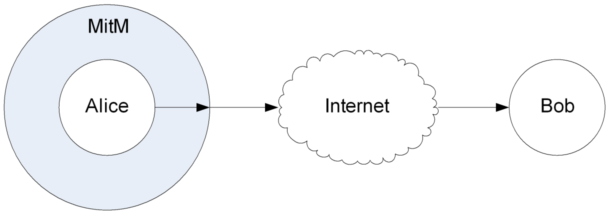

Figure 3: Scenario 1: MitM shields Alice.

sending a token $M_{bob} = (T, U, Y, R)$, which is a tuple containing the parameters defining the puzzle $P(T, U, Y)$, as well as the puzzle result $R(P)$. Alice now also constructs the puzzle P and checks whether $R(P)$ is really a solution to the puzzle, and also whether the token validity period T covers the current time. If the result is validated, the common session key is again derived by using the Diffie-Hellman approach. The session key may now be used for creating a VPN between Alice and Bob, and the ZRTP channel is established through this VPN, totally independent of the previously created session key.

3.1.1 Attack Analysis Scenario 1

An important point in our approach is given by the fact that the proposed scheme is *not* an authentication mechanism. Receiving a token from Bob within dt_c seconds does not mean that the identity of Bob is guaranteed 100%. Our proposed scheme is an enhancement of ZRTP, which is aimed at making a Mafia Attack much more difficult in case Alice and Bob do not know each other.

On the other hand, as was stated above, if Alice and Bob *know* each other, and the MitM knows about this relation, he is able to precompute tokens for every day of the next years for Bob. However, in this situation there is still ZRTP, which works well here. A real problem is given by a situation when Alice and Bob do not know each other and still the MitM is able to predict that Alice will call Bob in the future, and therefore precomputes tokens for Bob. In such a situation, our solution fails and the connection is solely protected by the security mechanisms of ZRTP. As a consequence our scheme is meant only as an enhancement to an existing system like ZRTP, not as an authentication mechanisms in itself. This is also the reason for creating a VPN tunnel and not using the derived session key for ZRTP, since in some cases a MitM might be able to interfere with our scheme, and using the key also for ZRTP would render the whole scheme to be endangered. Thus, our scheme is designed to be an additional independent security layer to ZRTP.

A MitM might also try to precompute tokens for every possible URI for every day of the near future. Given the needed computational power, this is clearly not possible using any kind of computer today.

Since the MitM is not able to compute the result of the puzzle within the required time, another possibility for the MitM is to exchange Bob's public key Y with its own public key Z in the token M_{bob}. However, in this case the problem P_M will be different than Bob's original problem P, and R is not a valid solution to P_M.

The MitM may also prevent the establishment of any communication, until the MitM has computed a suitable token. The computation starts as soon as Alice tries to communicate with Bob. The MitM catches the INVITE, does not forward it to Bob, but starts computing his own R. If the MitM commands the usage of an extremely powerful supercomputer, the result might be ready within several minutes. Afterwards, the MitM waits for the next call attempt from Alice to Bob, and is able to return its own token to Alice. It is obvious that in scenario 1, the MitM can always completely block any communication attempt Alice initiates. However, after the first call attempt Alice can detect that she has received no token from Bob within the critical time, and our scheme demands that in such a case, Alice must immediately switch to the second scheme called Random Call Delay (see Section 5).

An intrinsic property of our proposed scheme is the fact that during time period T (e.g., Sept. 1st, 2007, 00:00 am - 12:00 am) Bob will always reuse the same token, and thus the same public key Y within this period. However, since in scenario 1 we assume that the MitM shields Alice, and the scheme is mainly meant for situations when there is no relation between Alice and Bob, Bob as the *called person* is safe to reuse his public key Y when answering calls.

3.2 Scenario 2: MitM shields Bob

Figure 4 shows the second scenario that we investigate. In this scenario the MitM

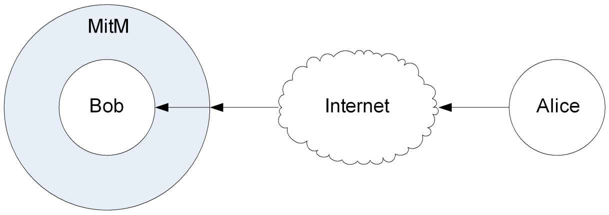

Figure 4: Scenario 2: MitM shields Bob.

shields Bob, who is called by Alice. Again we assume that there is no relation between Alice and Bob, and the MitM can not foresee that Alice will call Bob in the future.

Here we propose that Alice is required to attach her own token M_{alice} for this specific time period and her URI to the INVITE message, and that Bob will only answer to INVITE messages with attached token by sending his own token M_{bob}. Again Alice must receive this token M_{bob} from Bob within the critical time.

3.2.1 Attack Analysis Scenario 2

Again the MitM could simply prevent any communication between Alice and Bob until he has computed a token for Alice. Again Alice can detect that she was not able to establish a call with Bob and must switch to Random Call Delay.

It turns out that scenario 2 is much more subtle than scenario 1, since in this situation the MitM of course knows Bob and can easily precompute a token $\overline{M}_{bob}$ for Bob's URI, which he immediately sends back to Alice. Alice is satisfied and the ZRTP connection between her and the MitM can commence over the VPN between Alice and the MitM. However, the MitM must now in parallel start a conversation with Alice, and start to compute a token $\overline{M}_{alice}$ which contains Alice's URI, and which he needs for establishing a connection between himself and Bob. The result is a significant delay between starting the conversation with Alice, and being able to relay the call between Alice and Bob. If the MitM commands a powerful computer then he will be able to produce such a token after a few minutes (see Section 4). One important finding of our investigation of Mafia Attacks is the fact that they are extremely difficult. In fact we think that they are only possible if there is synchronous conversation between Alice and Bob, i.e., the MitM in realtime repeats everything he hears without thinking about it. As soon as the MitM is forced to produce meaningful content, i.e., pretend to be somebody else, or to relay the call with a delay of more than one second, the fraud will most certainly be detected. Of course, in reality, an attacker is more likely to need much longer than two minutes to compute a valid token, and thus again the overall result is that a Mafia Attack is much more difficult.

A final option for the MitM is now to precompute generic tokens for generic URIs, like john@bank.com. Upon receiving an INVITE from Alice, the MitM may select a best fitting generic token and immediately open a connection to Bob. Bob of course sees this URI, and must now decide whether this URI makes sense or not.

As overall result, scenario 2 is much more difficult to protect, but as stated before, the purpose of our approach is to make the life of the MitM more difficult, since in both scenarios there is no general solution yet found for protection against MitM attacks without a PKI.

3.3 Existing Computational Puzzles

Computational puzzles have been proposed to make protocols resistant to flooding attacks. The main idea is that the client has to proof he spent a certain amount of computational effort before the server dedicates any resources. An important property of a computational puzzle is that they are very expensive to solve, but it is comparatively cheap to verify the solution.

3.3.1 Hashcash

Hashcash [Back 2002] is a non-interactive cost function, which means that the client chooses its own challenge or random start value. The client solves the puzzles by finding a value x that satisfies the following equation:

$$Hash(I, N, x) = \overbrace{000\dots000}^{\text{the } k \text{ first bits of the hash}} \qquad \underbrace{Y}_{\text{rest of the hash bits}}$$

Here, $Hash$ is a cryptographic hash function, I represents the identity of client (like an email address or a SIP URI), and N is a nonce chosen freely by the client. Solving the puzzle requires the client to find a value x so that the first k bits of $Hash(I, N, x)$ are zero. The cost of solving the puzzles depends exponentially on k. The server can verify the correctness of a solution easily by carrying out a single hash computation.

The cost function used by Hashcash can easily be parallelized. If the attacker has n computers at his disposal, every single one can start searching for the solution beginning with different initial values. In contrast, the cost function we are proposing is designed to be very difficult to be parallelized.

While the goal of Hashcash is to reduce the amount of email spam, research presented in [Laurie and Clayton 2004] comes to the conclusion that in order to be effective, puzzles would have to be so difficult that a significant number of legitimate email senders would be unable to continue their current level of activity.

There is also an interactive version of Hashcash, where the nonce N is provided by the server in a first step. A variant of Hashcash has been proposed to reduce SIP spamming [Jennings 2007].

3.3.2 Memory Bound Computational Puzzle

An acknowledged problem of computational puzzles is that the processing time required to solve a puzzle can vary enormously on different hardware platforms. Work that might take 10 seconds on a 3 GHz Pentium could take an hour or more on a Palm Pilot. Dwork et al. [Dwork et al. 2003] have proposed puzzles that rely on addressing large amounts of random access memory. Since memory access speeds vary considerably less than CPU speeds, time for solving a puzzle is similar across different machines. Experimental results presented in the paper show only a factor of 4.6 between fastest and slowest platform for the memory bound approach. In comparison, Hashcash – a CPU bound method – shows a factor of 43. However, the presented memory bound puzzle is again a search through a search space, which can very effectively be parallelized.

3.3.3 Sequential Puzzles

A puzzle which is not parallelizable has been published in [Back 2000], being is based on time-lock puzzles [Rivest et al. 1996]. However, in order to validate the result a secret must be known. Since the knowledge of the secret enables an attacker to quickly compute the result, this scheme is unusable for our purposes.

3.4 Requirements for Cost Functions

The requirements for a cost function that is suitable for our application are the following:

1. It should be hard to parallelize the solving process. By that, even an attacker with a lot of resources (like a botnet or parallel computer) should not have a considerable advantage.

2. Verifying the solution of a puzzle should be inexpensive and not rely on a secret.

3. The amount of data that has to be exchanged should be low.

4. The time required for solving the puzzle should be as independent as possible from processor type and speed.

As a matter of fact, none of the existing computational puzzles satisfy all conditions. Puzzles relying on searches through large search spaces can easily be parallelized. An attacker commanding thousands of computers would be able to compute the results quickly. The sole non-parallelizable puzzles relies on a secret for verification, which does not make sense in our scheme.

We thus propose to use a type of mathematical problem which is important for many applications of computational science and which has consequently been investigated extensively in the past, namely the numerical solution of *linear eigenproblems*.

4 Eigenvectors

In the following the standard computer arithmetic IEEE 754-1985 in double precision is assumed.[3] We consider eigenproblems which are defined by a real $N \times N$ matrix M as the basis for a computational puzzle. The puzzle is to compute a vector $x \in \mathbb{R}^N$ and a corresponding real scalar λ, such that the following equation is satisfied:

$$Mx = \lambda x. \tag{1}$$

The tuple (λ, x) is then called *eigenpair* of M. For general matrices there is no guarantee that the problem (1) has real solutions. However, it is known that for real *symmetric* M, there are exactly N linearly independent eigenvectors $x_i \in \mathbb{R}^N$. Thus, for our application context we only consider real symmetric matrices.

4.1 Computing Eigenvalues and Eigenvectors

Computation of eigenvalues is necessarily an iterative process since the eigenvalues are the roots of the characteristic polynomial. Many methods are available for solving problems of the type (1). Important distinctions among existing methods have to be made depending on whether the eigenanalysis of a *dense* or a *sparse* matrix is desired.

4.1.1 Reduction-Based Methods

Computing eigenvalues and eigenvectors of a *dense* symmetric matrix usually involves a preprocessing step in order to reduce the complexity of the eigenanalysis, because it would be prohibitively expensive to work with the original full matrix. The given matrix is reduced to similar condensed form; *tridiagonal* form in the symmetric case. In a tridiagonal matrix the only non-zero matrix

[3] http://grouper.ieee.org/groups/754/

elements are located in the main diagonal, in the first superdiagonal, and in the first subdiagonal.

In contrast to this reduction operation, which is a finite, non-iterative process, the actual computation of eigenvalues and eigenvectors is an iterative process (as mentioned before). Nevertheless, methods which perform a preprocessing step to condensed form are often called "direct" methods in the literature. In the following, they will be called *reduction-based* methods.

4.1.2 Subspace Methods

If the given symmetric matrix is *sparse*, then a reduction step is usually not advisable because it produces fill-in. In this case, methods based on projections onto small subspaces, for instance, the Lanczos method [Lanczos 1950] (see, for example, also Paige [Paige 1972],[Paige 1976]), or the Jacobi-Davidson method [Sleijpen and van der Vorst 1996] tend to be much more efficient (see also [Van der Vorst and Golub 1996]). Often the terminology "iterative methods" is used for methods of this type. Since *every* method for computing eigenvalues needs to be iterative (at least partly), we use the term *subspace* methods instead.

4.1.3 Relevant Algorithms and Software

In this section, a brief overview of standard algorithms for solving real symmetric eigenproblems which are most relevant for our application context is given. Much more comprehensive coverage of various algorithms for solving symmetric eigenproblems can be found in various standard text books, for example, [Golub and Van Loan 1996] or [Parlett 1997]. Many practical issues related to numerical software for eigenvalue problems are discussed in [Demmel 1997] and in [Bai et al. 2000].

In the area of reduction-based methods, the reduction step itself has to be followed by a solver for the symmetric tridiagonal eigenproblem. Various methods for such problems have been designed, among the most important ones being the tridiagonal divide-and-conquer (*TD&C*) method [Gu and Eisenstat 1994], [Gu and Eisenstat 1995] and the *MRRR*-method:

- [Dhillon 1997], [Parlett and Dhillon 1997], [Parlett and Dhillon 2000],
- [Dhillon and Parlett 2004b], [Dhillon and Parlett 2004a],
- [Dhillon et al. 2005], and [Dhillon et al. 2004].

The latter, which is considered the "Holy Grail" for symmetric tridiagonal eigenproblems, represents the latest developments in this area and is the most modern algorithm available. Currently, for most situations one of these two algorithms can be expected to achieve the best performance for symmetric tridiagonal eigenproblems.

Subspace methods in their standard form usually compute extreme eigenpairs at the lower or upper end of the spectrum. If *interior* parts of the spectrum are to be computed, a suitable starting vector or some form of estimate of an interior eigenvalue for a shift-and-invert approach is required. This also holds for the power method, for inverse iteration, and for its very fast converging more advanced relative, Rayleigh quotient iteration, which almost always exhibits global cubic convergence [Parlett 1997, Golub and Van Loan 1996].

As outlined in Section 4.4, we design our computational puzzle such that (i) the matrix M is dense, (ii) the computation of a specified *interior* eigenpair is required, and (iii) no prior information about this eigenpair is available (neither an approximation of the eigenvalue nor a good starting vector for subspace methods). Consequently, there is no clear advantage of subspace methods in terms of computational efficiency, and therefore the following discussion concentrates on the currently best reduction-based methods.

State-of-the-art high quality software for numerically solving large-scale symmetric eigenproblems is available in various libraries, many of them available at `http://www.netlib.org`. For reduction-based methods, the de-facto standard libraries are LAPACK [Anderson et al. 1999] for sequential and shared memory computers, and SCALAPACK [Blackford et al. 1997] for distributed memory computers.

4.2 Computational Complexity

As reviewed in the following, the computational complexity of computing an interior eigenpair of a dense symmetric matrix M is in general $O(N^3)$ floating-point operations (flops). However, given a pair (λ, x), it only takes $O(N^2)$ (or even only $O(N)$, see Section 4.4.1) floating-point operations to decide whether this pair is a sufficiently accurate approximation of an eigenpair of M (via computation of the residual). This property that computing a solution requires asymptotically a higher order effort than accepting or rejecting a candidate solution makes the problem (1) an interesting computational puzzle for our application context.

4.2.1 Tridiagonalization

As a first step in reduction-based methods, the matrix M is reduced to tridiagonal form. For a dense matrix M this preprocessing step is very significant in terms of arithmetic complexity and even dominates the work required for solving the standard eigenproblem if fewer than N eigenpairs are to be computed.

In summary, Householder-based tridiagonalization of a dense symmetric matrix M requires $4N^3/3 + O(N^2)$ floating-point operations for constructing and applying the Householder reflections when symmetry is exploited in the update operation [Golub and Van Loan 1996]. If the accumulated transformation matrix is required explicitly then it can be formed with an additional $4N^3/3+O(N^2)$ flops [Golub and Van Loan 1996].

On a single processor, this tridiagonalization part is one of the components responsible for the N^3 scaling behavior of the execution time with the problem size N, which can be observed in most circumstances.

4.2.2 Tridiagonal Problem

One eigenvector of a symmetric tridiagonal matrix T can be computed with about $16N$ flops assuming that no pivoting is required in the factorization of T [Golub and Van Loan 1996] and that at most two iterations suffice to reach the desired accuracy (cf. Ipsen [Ipsen 1997]). If several eigenvectors corresponding to very close eigenvalues are sought, then special care has to be taken in inverse iteration to avoid a loss of orthogonality in the computed eigenvectors. In fact, in the worst case reorthogonalization can cause standard inverse iteration to be an $O(N^3)$ process, even for tridiagonal matrices. This problem is one of the aspects where the most recent development for tridiagonal eigenproblems, the MRRR-algorithm mentioned before, made substantial progress.

Currently available versions of the TD&C-algorithm are highly competitive if all N eigenpairs are to be computed, but they are less attractive when only a single eigenpair has to be computed, as in our setup.

4.3 Parallelization

TD&C has attractive parallelization properties [Tisseur and Dongarra 1999], and ScaLapack contains a parallel code for this method. A parallel implementation of the MRRR algorithm, which promises substantial improvements for the cases when only parts of the spectrum are to be computed and also a potentially better scaling behavior, is currently under development and can be expected to be included into ScaLapack soon.

Comprehensive investigations documented in the literature have shown that reduction-based methods for dense symmetric eigenproblems are not easy to parallelize [Ward et al. 2005, Ward 2006]. It is in general difficult to achieve good scaling behavior over large numbers of processors and in massively parallel environments. However, important progress has been made recently with the development of parallel codes for the MRRR-algorithm [Sunderland 2006] and in the parallelization [Bai and Ward 2007] of variants of the divide-and-conquer approach.

Very generally speaking, the following statements hold with respect to parallelization of reduction-based methods for dense symmetric eigenproblems of the type (1).

1. For fixed problem size N and growing processor number p, the execution times first tend to decrease rapidly (high speed-up), then the curve tends to flatten out (low speed-up), reaching a minimum for some processor number p^*, and finally the execution times tend to increase again (slow-down). This implies that for a fixed problem size the execution time for current state-of-the-art eigensolvers *cannot be reduced arbitrarily* by increasing the parallelism.

2. Obviously, the processor number p^* for which the execution time is minimum depends on many different factors, including the problem size N, hardware properties (interconnect, communication speed, ratio of communication to computation speed, etc.). Roughly speaking, for N below 10000, p^* tends to be significantly below 1000 for current state-of-the-art hardware (cf. [Ward et al. 2005, Ward 2006, Sunderland 2006]).

3. Increasing the problem size N also tends to improve the parallelization properties and thus to increase p^*.

Although the parallelization of eigensolvers is easier on shared memory machines because of the faster communication, the basic performance patterns are similar. This has been confirmed in a series of experiments which we performed on *Goedel*, a Sun Fire v40z server with 4 dual-core AMD 875 2.2 GHz processors and 24 GB of memory. The runtimes for computing one eigenpair using the latest version of the parallel MRRR-code (to be included into ScaLapack soon) are shown in Figure 5. Due to the fact that only 8 processors are available on this machine, it was not possible to experimentally determine the processor number p^* for most problem sizes.

4.4 Design of a New Computational Puzzle

The previous brief summary of the state-of-the-art in dense symmetric eigensolvers indicates the major guidelines for designing a computational puzzle for the context discussed here. The central objective is to guarantee that some minimum time $T_{\min}$ is required to compute the required eigenpair of a dense symmetric matrix M accurately enough. Additionally, since the solution of the puzzle must be sent over the network, its size should be as small as possible. In the following we investigate the order of magnitude of the number of floating-point operations necessary for computing the solution of a puzzle (determining the time it takes to compute the result), and also the order of magnitude of the size of the solution (determining the number of bytes that must be sent over the network).

From the above it follows that

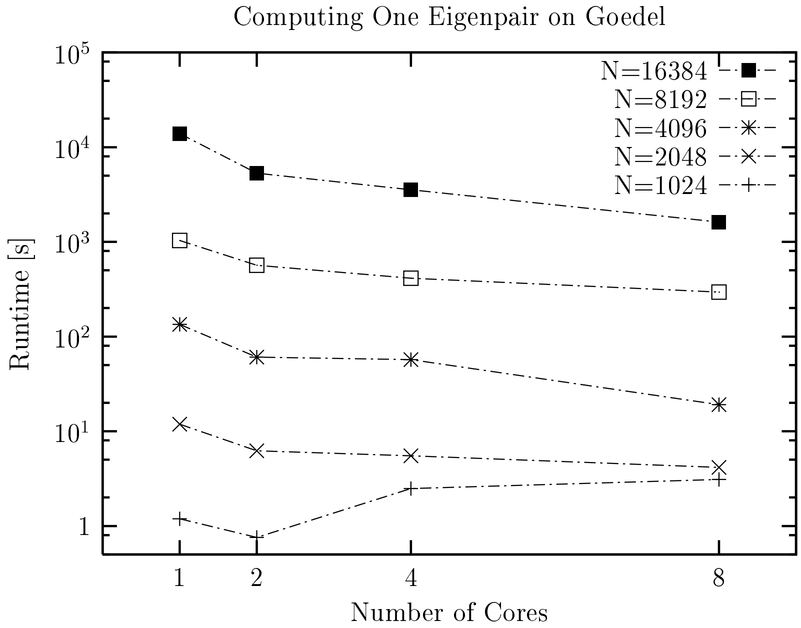

Figure 5: Runtimes of parallel MRRR code on a 4-processor dual-core SMP machine.

1. N cannot be too small in order to exceed $T_{\min}$ when sequential processing is used; and
2. N should not be too large since it determines the size of the solution of the puzzle. Also, the degree of reasonable parallelization depends on N, the larger N, the better the problem can be parallelized, which is undesirable in our scenario.

One way to achieve both objectives is to construct a computational puzzle which consists of solving a sequence of K eigenproblems P_i, $1 \leq i \leq K$. Problem P_i is defined via a matrix M_i, and is solved by finding at least one *interior* eigenpair $R_i = (x_i, \lambda_i)$ with $M_i x_i = \lambda_i x_i$. The size of each subproblem can be kept relatively small, the strictly sequential structure restricts the benefits of parallel processing, and the overall time required for solving the puzzle can be adjusted via the parameter K.

The initial eigenproblem P_1 is defined by the input data $H_1 = Hash(T, U, Y)$ such that H_1 is used as a seed for constructing a random $N \times N$ real symmetric matrix M_1. Each eigenproblem $P_i,\ 1 < i \leq K$ is then defined by a random matrix

M_i, which is constructed by setting the seed of a random number generator to

$$H_i = Hash(R_{i-1}, H_1, \ldots, H_{i-1}), \tag{2}$$

i. e., a hash over the previous solution and all previous hashes. The solution to the computational puzzle is the ordered set of solutions $R = (R_1, R_2, \ldots, R_K)$.

Result R of a token $M_{bob} = (T, U, Y, R)$ contains K eigenvectors of length N, each component being a double (8 bytes). The necessary amount of data to be transferred is thus $O(KN)$ bytes. The matrices themselves do not have to be transmitted, since they can be constructed from the token itself. If Bob sends such a token to Alice, Alice will first have to set up the matrices M_i and then test whether $R_i = (x_i, \lambda_i)$ is a valid solution by multiplying x_i on M_i. The complexity of constructing a full $N \times N$ matrix is of order $O(N^2)$ flops, vector-matrix multiplication is of order $O(N^2)$ flops, and thus the computational complexity of validating R is of order $O(KN^2)$ flops.

4.4.1 Reduction of the Complexity

However, it turns out that both sending and validating the result R can be improved significantly. A straightforward improvement is to require that for validating the solution $R_i = (x_i, \lambda_i)$ not all vector components must be tested, but only a fixed number, for instance, 20. This means that Alice multiplies x_i only onto 20 randomly chosen rows $M_i(k, :)$ of matrix M_i, thus reducing the vector-matrix multiplication to $O(N)$ floating-point operations. The result is validated if

$$M_i(k, :) \cdot x_i = \lambda_i x_i(k) \text{ for all chosen values of } k,$$

i.e., the scalar product of matrix row $M_i(k, :)$ and eigenvector x_i results in the k-th component of x_i times λ_i. There is currently no known procedure for identifying vectors, where a certain fraction of components would yield the same result.

For reducing the overall validation complexity, we also have to reduce the matrix construction to $O(N)$ operations. This can, for example, be achieved by constructing the real symmetric matrix M_i strictly element-wise, for example, by setting its elements $M_i(k, l)$ to

$$M_i(k, l) = \begin{cases} F(Hash(H_i, k, l)) & \text{if } 1 \le k \le N, k \le l \le N \\ M_i(l, k) & \text{else} \end{cases}.$$

Here, $F(I)$ denotes a function that turns a large random positive integer I into a real value from a predefined value range. Hence, only those rows $M_i(k, :)$ have to be created which are needed for the above mentioned scalar products $M_i(k, :) \cdot x_i$. This way, the computing effort required for validating a single result R_i can be decreased to $O(N)$ flops instead of $O(N^2)$, and validating R thus becomes $O(KN)$ flops instead of $O(KN^2)$.

The second improvement is given by the fact that it is not necessary to validate all solutions R_i, but rather only a subset. We thus propose that instead of sending the solution R, Bob initially only sends $\overline{R} = (H_1, H_2, \ldots, H_{K+1})$ which actually defines the matrices M_i. Bob thus *commits* himself to the results R_i without sending them. Upon receiving the token $\overline{M}_{bob} = (T, U, Y, \overline{R})$, Alice then asks Bob to send a randomly selected subset of the R_i, for instance R_2, R_{10}, and R_{K-1}. Alice can then set up matrices M_2, M_{10} and M_{K-1} (or only a small number of rows), validate the R_2, R_{10}, and R_{K-1}, and test whether

$$\begin{aligned}
H_1 &= Hash(T, U, Y) \\
H_3 &= Hash(R_2, H_1, H_2), \\
H_{11} &= Hash(R_{10}, H_1, \ldots, H_{10}), \quad \text{and} \\
H_K &= Hash(R_{K-1}, H_1, \ldots, H_{K-1}).
\end{aligned}$$

The more results R_i Alice requests, the more data must be sent over the network, but also the more reliable is the result. Furthermore, due to the inclusion of the previous hashes into (2), in the above scheme it is also proven that the results R_2, R_{10} and R_{K-1} were computed in sequence and not in parallel (e.g., in order to create matrix M_{10}, the hash H_3 and thus the result R_2 must be known).

As a result, the time for solving our puzzle depends on the number of floating-point operations, which are of $O(KN^3)$, while sending the result and validating costs are at most $O(KN)$ (bytes and flops), or, if reduced reliability is acceptable, only $O(N)$ (bytes and flops).

4.4.2 Determining the Optimal Puzzle

Given a certain critical time dt_c (which depends on the properties of the connection available for the call), we need to identify the range of all tuples (N, K), for which the minimum time required to solve the sequence of K symmetric eigenproblems of size $N \times N$ each as outlined above is greater than dt_c. Within this parameter range, we then need to determine those puzzles, for which the amount of data to be transferred $(N \times K)$ is minimum. In a more compact formulation:

$$\min_{N,K} N \times K \quad \text{for which} \quad T(N, K) > dt_c.$$

This basic concept allows for the concrete definition of an "optimal" computational puzzle, which is currently under development and will be discussed in detail in a forthcoming publication.

5 Random Call Delay

The second mechanism we propose is comparable to the one of Robin and Schwartz [Robin and Schwartz 2006]. This approach is based on the hypothesis that a MitM attack on ZRTP is impossible if the caller or callee are not synchronously starting the call at the same time, but instead there is a significant delay. Such a delay could mean that the caller first starts a call with the MitM, and then after some time, the MitM starts the call with the callee. They therefore propose that both caller Alice and callee Bob setup a call at time t and immediately negotiate a value N, where N is the maximum amount of time that both are willing to wait for the call to start. The VoIP clients then start a timer and create a hash $Hash(K_{AB})$ over the secret Diffie-Hellman Key K_{AB}. Since this number should be uniformly distributed over its range $[0, H_{max}]$, it is then transformed into the range $[0, N]$ by

$$dt_{AB} = N\frac{Hash(K_{AB})}{H_{max}}. \tag{3}$$

Both Alice and Bob are then advised to start talking about their issues at time

$$t + dt_{AB} + r, |r| < \tau. \tag{4}$$

Because of network latency and SIP proxy delay, there must be some kind of tolerance τ for coping with network latency variation.

In our opinion this scheme involves too much attention of the involved persons, keeping them busy at the phone for the whole duration of the timer. Even more so, since in order to work efficiently, the maximum waiting time N should be quite large, in the order of minutes. Furthermore, the whole scheme depends on Alice and Bob, and whether they strictly adhere to the described procedure.

We therefore propose a variant of the above scheme, which works as follows. Similar to the scheme of Robin and Schwartz, at time t the VoIP clients negotiate a quasi random number dt_{AB} depending on the secret key K_{AB} via (3). Then the clients immediately cancel the call. This can be done in a way that the callee Bob is not even notified that he has been called. Alice must then wait until her client automatically redials the callee after the timer has elapsed, but is free to do something else. At time $t+dt_{AB}$ Alice's client calls Bob's client, which accepts the call only if it happens right at the negotiated time (4), thus summoning Bob who only from this time on is involved. In case Bob answers the call, Alice's client rings and summons Alice's attention.

In the second call, however, the media encryption must be based on the secret session key K_{AB} that has been negotiated in the first call, since this key is actually validated by the call delay scheme. Again we propose to tunnel the following ZRTP session through a VPN encrypted with this key K_{AB}.

5.1 Attack Analysis Random Call Delay

The MitM here must intercept the first call attempt from Alice at time t, thus generating a dt_{AM}. Alice's client will again call at $t + dt_{AM}$ and expect to establish the call.

The MitM now has two choices. First he may continuously call Bob at times $t = t_1, t_2, \ldots t_n < t + dt_{AM} + \tau$, thus creating a sequence of delay times

$$dt_{MB,1}, dt_{MB,2}, \ldots, dt_{MB,n}$$

caused by the respective keys between the MitM and Bob. The MitM stops as soon as a delay time collision is detected:

$$|(t + dt_{AM}) - (t_i + dt_{MB,i})| < 2\tau.$$

For reasonably small values of τ, the collision probability for one call i is $4\tau/N$, and the probability that at least one collision out of n calls is created, is therefore $1 - (1 - 4\tau/N)^n$. Such a sequence of calls, however, can easily be detected by Bob's client, for example by detecting that too many call attempts are made per time unit.

The final option left to the MitM is thus to call Bob once, and later start the real call at a totally different time than $t + dt_{AM}$. However, due to the above hypothesis, relaying such a call is hardly possible.

As a consequence, by using Random Call Delay, we achieve the same level of security as described in [Robin and Schwartz 2006], but only keep the caller waiting, who however in the meantime is free to do something else. Because of this, callers may accept longer waiting times (i.e., a larger N), which automatically increases the level of security due to decreased collision probabilities. The callee on the other side is only included once the real call is established.

5.2 Combining Tokens and Call Delay

As was stated above, in this paper we propose to combine both the above described challenge-response scheme, plus the new Random Call Delay. When dialing Bob's URI, the client should ask Alice whether she knows Bob and has ever had contact with him before. If Alice denies, the client invokes the computational puzzle scheme. This allows Alice to skip the waiting time if Bob is able to present a valid token in time. For this, however, there is only one try. If Alice does not receive a valid token within the critical time, the client marks Bob's URI as unsafe, and Alice should from this time on only use Random Call Delay to establish a first call (from the second call on, the key continuity of ZRTP prevents further attacks).

In case Alice knows Bob, or she has not received a valid token within the critical time, Alice's client only offers Random Call Delay, and if Alice wants to have a secure call she must be prepared to wait a few minutes.

Preventing the first call attempt of course can be caused by two ways. First, Bob's VoIP client might not be registered currently, or any other part of the VoIP infrastructure might be currently unavailable. Second, the MitM might simply block every first call to any URI he does not know. The first situation can be improved by using some kind of highly available VoIP system, including highly available infrastructure and clients. In this case, a call which fails at the first time may arouse suspicion. The second situation of course cannot be prevented in the scenarios under consideration, but Random Call Delay as an automatic fallback remains.

6 Conclusion

In this paper we have shown three innovations. The first innovation is a new challenge-response scheme for VoIP systems without PKI, as can be created by ZRTP. In order to pass the challenge, both caller and callee must present the result of a computational puzzle within a critical time. This can be done only by precomputing it, an attacker in the media path would not be able to do this in time. The scheme works best in situations, where ZRTP is vulnerable most, i.e., if both caller and callee do not know each other, and a MitM could launch a Mafia Attack.

In the absence of a 100% reliable network, the challenge-response could be efficiently blocked by a MitM. In such a situation, we propose our second innovation, which delays the call for a random time depending on the shared Diffie-Hellman session key. The caller would call only after this time has passed by, and the callee only accepts such a call at this time. Due to our approach, only the caller experiences the delay, while the callee does not.

The third innovation is a new computational puzzle which computes eigenvectors of large matrices. This puzzle does not rely on shared secrets for validation, but still is difficult to be parallelized for reasonably sized matrices. We have further shown that for validating the result, only a small subset of the vector components needs to be validated. Furthermore, by using hash commitment, only a subset of the eigenvectors must be transmitted.

References

[Anderson et al. 1999] Anderson, E., Bai, Z., Bischof, C. H., Blackford, S., Demmel, J. W., Dongarra, J. J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D. C.: LAPACK *Users' Guide*. SIAM Press, Philadelphia, PA, 3rd edition, 1999.

[Aura et al. 2001] Aura, T., Nikander, P., Leiwo, J.: DOS-resistant Authentication with Client Puzzles, 2001, `http://research.microsoft.com/users/tuomaura/Publications/aura-nikander-leiwo-protocols00.pdf`

[Back 2000] Back, A.: Hashcash - amortizable publicly auditable cost functions. Early draft of paper, 2000.

[Back 2002] Back, A.: Hashcash - A denial of Service Counter-Measure, Tech Report, Aug 2002

[Bai and Ward 2007] Bai, Y., Ward, R. C.: A parallel symmetric block-tridiagonal divide-and-conquer algorithm. *ACM Trans. Math. Softw.*, 33(4):25, 2007.

[Bai et al. 2000] Bai, Z., Demmel, J., Dongarra, J. J., Ruhe, A., van der Vorst, H., editors: *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*. SIAM Press, Philadelphia, PA, 2000.

[Blackford et al. 1997] Blackford, L. S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J. W., Dhillon, I., Dongarra, J. J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., Whaley, R. C.: ScaLapack *Users' Guide*. SIAM Press, Philadelphia, PA, 1997.

[Demmel 1997] Demmel, J. W.: *Applied Numerical Linear Algebra*. SIAM Press, Philadelphia, PA, 1997.

[Dhillon 1997] Dhillon, I. S.: *A New $O(n^2)$ Algorithm for the Symmetric Tridiagonal Eigenvalue/Eigenvector Problem*. PhD thesis, Computer Science Division (EECS), University of California at Berkeley, 1997.

[Dhillon and Parlett 2004a] Dhillon, I. S., Parlett, B. N.: Multiple representations to compute orthogonal eigenvectors of symmetric tridiagonal matrices. *Linear Algebra Appl.*, 387:1–28, 2004.

[Dhillon and Parlett 2004b] Dhillon, I. S., Parlett, B. N.: Orthogonal eigenvectors and relative gaps. 25:858–899, 2004.

[Dhillon et al. 2004] Dhillon, I. S., Parlett, B. N., Vömel, C.: The design and implementation of the MRRR algorithm. Lapack Working Note 162, University of California, Berkeley, Berkeley, CA, 2004.

[Dhillon et al. 2005] Dhillon, I. S., Parlett, B. N., Vömel, C.: Glued matrices and the MRRR algorithm. *SIAM J. Sci. Comput.*, 27(2):496–510, 2005.

[Dwork et al. 2003] Dwork, C., Goldberg, A., Naor, M.: On Memory-Bound Functions for Fighting Spam. in D. Boneh (Ed.): Advances in Cryptology CRYPTO 2003, LNCS 2729, Springer Verlag 2003, pp. 426-444.

[Golub and Van Loan 1996] Golub, G. H., Van Loan, C. F.: *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, 3rd edition, 1996.

[Gu and Eisenstat 1994] Gu, M., Eisenstat, S. C.: A stable and efficient algorithm for the rank-one modification of the symmetric eigenproblem. *SIAM J. Matrix Anal. Appl.*, 15:1266–1276, 1994.

[Gu and Eisenstat 1995] Gu, M., Eisenstat, S. C.: A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem. *SIAM J. Matrix Anal. Appl.*, 16:172–191, 1995.

[Ipsen 1997] Ipsen, I. C. F.: Computing an Eigenvector With Inverse Iteration, SIAM Rev. 1997-39, pp. 254–291.

[Jennings 2007] Jennings, C.: Computational Puzzles for SPAM Reduction in SIP, July 2007, `http://tools.ietf.org/html/draft-jennings-sip-hashcash`

[Lanczos 1950] Lanczos, C.: An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *J. Res. Nat. Bur. Stand.*, 45:255–282, 1950.

[Laurie and Clayton 2004] Laurie, B., Clayton, R.: "Proof-of-Work" Proves Not to Work, May 2004

[Paige 1972] Paige, C. C.: Computational variants of the Lanczos method for the eigenproblem. *J. Inst. Math. Appl.*, 10:373–381, 1972.

[Paige 1976] Paige, C. C.: Error analysis of the Lanczos algorithm for tridiagonalizing a symmetric matrix. *J. Inst. Math. Appl.*, 18:341–349, 1976.

[Parlett 1997] Parlett, B. N.: *The Symmetric Eigenvalue Problem.* SIAM Press, Philadelphia, PA, 1997.

[Parlett and Dhillon 1997] Parlett, B. N., Dhillon, I. S.: Fernando's solution to Wilkinson's problem: An application of double factorization. *Linear Algebra Appl.*, 267:247–279, 1997.

[Parlett and Dhillon 2000] Parlett, B. N., Dhillon, I. S.: Relatively robust representations of symmetric tridiagonals. *Linear Algebra Appl.*, 309:121–151, 2000.

[Rivest et al. 1996] Rivest, R. L., Shamir, A., Wagner, D. A.: Time-lock puzzles and timed-release crypto. Technical Report MIT/LCS/TR-684, 1996. `http://theory.lcs.mit.edu/$\sim$rivest/publications.html`

[Robin and Schwartz 2006] Robin, J., Schwartz, A.: Analysis of ZRTP, Protocol of final project. `http://www.stanford.edu/class/cs259/projects/project05/05-Writeup.pdf`

[Sleijpen and van der Vorst 1996] Sleijpen, G. L. G., van der Vorst, H. A.: A Jacobi-Davidson iteration method for linear eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 17:401–425, 1996.

[Sunderland 2006] Sunderland, A.: Performance of a new parallel eigensolver pdsyevr on hpcx. Technical Report HPCxTR0610, Computational Science and Engineering Department, CCLRC Daresbury Laboratory, Warrington, UK, 2006.

[Tisseur and Dongarra 1999] Tisseur, F., Dongarra, J.: A parallel divide and conquer algorithm for the symmetric eigenvalue problem on distributed memory architectures. *SIAM J. Sci. Comput.*, 20:2223–2236, 1999.

[Van der Vorst and Golub 1996] van der Vorst, H. A., Golub, G. H.: 150 years old and still alive: Eigenproblems. Technical Report SCCM-96-11, SCCM Program, Department of Computer Science, Stanford University, Stanford CA, 1996.

[Ward 2006] Ward, R. C.: Performance of parallel eigensolvers on electronic structure calculations ii. Technical Report UT-CS-06-572, Department of Computer Science, University of Tennessee, Knoxville, TN, September 2006.

[Ward et al. 2005] Ward, R. C., Bai, Y., Pratt, J.: Performance of parallel eigensolvers on electronic structure calculations. Technical Report UT-CS-05-560, Department of Computer Science, University of Tennessee, Knoxville, TN, February 2005.