A Protocol for Secure Electronic Remote Voting

Edoardo Biagioni, Yingfei Dong, W. Wesley Peterson, and Kazuo Sugihara
University of Hawaii at Mānoa
esb@hawaii.edu

***Abstract*—We have developed Distributed, Voter Verified, Secret Ballot Voting (DVVSB) [1], a potentially remote voting system that allows a voter to directly verify that their votes are in the final database of tallied votes. DVVSB meets key requirements of voting systems, such as voter anonymity and prevention of various vote insertion attacks, e.g., voting more than once or voting for others. DVVSB is also resistant to malicious interferences including several common types of Denial of Service (DoS) attacks. We summarize how DVVSB allows voters to cast and verify anonymous ballots, then focus on the networking and distributed computing techniques used to meet these security goals.**

I. Introduction and Related Work

The traditional paper voting process has many advantages and many disadvantages. Chief among the advantages is that many mechanisms have been developed to track paper ballots and to verify, with high likelihood, what the voter's intent is on a paper ballot [2]. Among the disadvantages of paper ballots are expense, physical vulnerability, including ballot box stuffing, and the small but sometimes significant number of votes for which the voter's intent cannot reliably be determined. Good examples are offered by Shamos [3] on pages 15-17. A more recent example was the 2008 senatorial election in the U.S. state of Minnesota [4].

Electronic voting machines would seem to address many of these issues, especially if backed by Voter Verifiable Paper Audit Trails (VVPATs). However, even the paper audit trail is produced by software and is vulnerable to tampering by insiders and well-organized attackers, as well as subject to running out of ink. NIST has collected a number of papers documenting such issues [5].

A number of systems, both paper and electronic, have been proposed that would allow voters to verify that their votes were "cast as intended", and to probabilistically be assured that there votes were "counted as cast" [6]. Voters cannot directly verify that their individual votes are counted as cast and as intended. This is seen as a strength of these systems, as it puts up a barrier vote coercion and vote buying.

However, these schemes have drawbacks. The resistance to coercion and vote buying is weak whenever an election allows write-in voting. Elections that allow remote voting, including postal (absentee) ballots, are vulnerable to vote buying or coercion no matter what technology is used. Further, voters using these so-called "end-to-end" voting systems [7][8][9] have to place their trust in a mechanism too complicated for most voters to understand, or even in a database that must be trusted to store correct ballot interpretations.

In contrast, in Distributed Voter Verified Secret Ballot or DVVSB [1], each voter has a unique ballot ID. This ballot ID is attached to the ballot throughout the process, and is only known to the voter. Once the election is over, the database of all ballots can be published. Each voter can then verify that the ballot with the unique ballot ID is indeed in the database, and counted correctly corresponding to the voter's intent. Further, anyone can compare the number of ballots in the database to the number of ballots submitted for certification, and to the number of registered voters, verifying that the counting has been performed correctly. This simple operation corresponds to an audit in paper- or DRE-based electronic elections, and can be carried out by anyone with access to the Web. This is a stronger and more easily understood end-to-end guarantee than offered by any other proposed system.

With DVVSB any problems with the software or any one misbehaving individual can be detected if it affects the outcome of the election, and can be detected by the individual voters.

Voters in DVVSB are anonymous as long as they do not divulge their ballot ID. Should a voter need to resist coercion or fool a vote buyer, they can obtain a completely valid ballot ID from the database of votes cast.

As the name suggests, DVVSB is suitable for distributed voting, for example, as a computerized replacement for the current practice of mail-in absentee ballots. DVVSB is also suitable for electronic voting machines in polling places, where one VTS can be local, and additional VTSs can be kept at the central precinct office. Any network disruptions during the election then would not affect the local recording of votes, and if the signing servers for the polling station were available locally as well, network availability becomes a convenience to facilitate tallying rather than a requirement.

A. Overview of DVVSB

This paper focuses on the networking protocol supporting DVVSB, and on the properties of this protocol that make it resistant to man-in-the middle and Denial of Service (DoS) attacks. We begin the description with a summary of DVVSB, while the details appear elsewhere [1]. The notation used in this paper is summarized in Table I.

The fundamental strategy of DVVSB has been presented before [10][11], and is based on Chaum's blind signatures [12].

In short, the voter uses a computer, the *voting client*, to enter the vote. This voting client accepts a ballot ID from the voter, concatenating enough random bits to effectively guarantee uniqueness of this ballot ID. The voter's ballot selection b

TABLE I
NOTATION

symbol	meaning
BAS	Ballot Authentication Server, blindly signs ballots
VTS	Vote Tallying Server, records votes
b	voter ballot choices
bts	ballot to sign: ballot ID with voter ballot choices
k	random blinding factor for bts
$bts*$	blinded version of bts, $bts* = bts \times k^e \bmod n$
n	modulo for an RSA public key
e	RSA public key for any server, always $65,537$
d	RSA secret key for a server
s	RSA signature for a bts
N	number of registered voters
N_{sig}	number of signatures needed to certify a ballot
m	maximum number of corrupted BASs
$aesk$	AES256 key for submitting the ballot to a tally server
a	encrypted version of nonce and AES-256 key
RSA	Rivest-Shamir-Adleman public-key algorithm [13]
$RSA2048$	RSA using $2,048$-bit keys
$AES256$	Advanced Encryption Standard [14] using 256-bit keys
DoS	Denial of Service attack

and the ballot ID together form the ballot to sign, bts. The only way to link a ballot ID to a specific voter is for someone to eavesdrop (perhaps electronically) during the voting process and record the ballot ID selected by the voter and the voting client. Without this eavesdropping, ballots are anonymous.

The voting client blinds bts so that it cannot be decrypted by others, and sends it to one or more Ballot Authentication Servers (BASs), each of which verifies the voter's credentials and blindly signs the ballot. The server then returns the blinded signature to the voting client. This signature confirms that the voter is entitled to vote and has not voted before, and commits the voter to this ballot. The blinding ensures that the BAS cannot tell who the voter is voting for, preserving the secrecy of the ballot.

Only the client can unblind the signature and verify it. The voting client then sends the unblinded ballot and unblinded signature(s) to one or more Vote Tallying Servers (VTSs), each of which verifies the signature and records the vote. The VTSs can read the ballot, but do not know the identity of the voter.

Once a vote is validly signed, it is a legitimate, anonymous vote. It can be submitted to a variety of organizations as well as to the official VTSs, and can be resubmitted until it is recorded in the final tally.

The option of multiple authentication and tallying servers is an innovation of DVVSB. A single BAS could, at the end of the election, fraudulently sign votes for registered voters that have not yet voted. Multiple BASs prevent this problem as long as less than $1/3$ of the BASs are colluding. Multiple VTSs can independently record submitted votes, making it harder for a vote to be lost – although a lost vote can be detected, it is better not to lose votes.

B. DVVSB Resistance to Attacks

The main focus of this paper is the design of the communication system to protect DVVSB voting against a number of attacks. Specifically, we have designed the system to be secure against man-in-the-middle attacks, and to be as resistant as possible to denial of service (DoS) attacks.

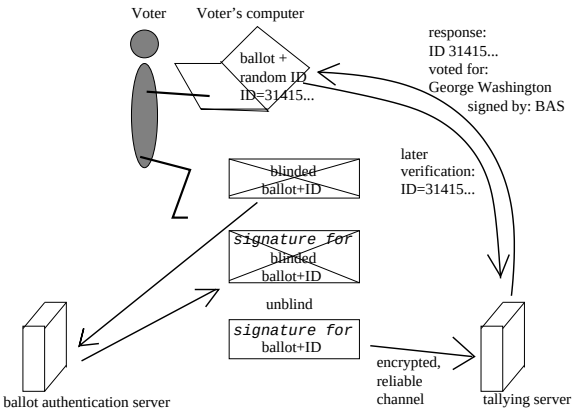

Fig. 1. Exchanges in the DVVSB Protocol. The final verification is optional.

In a man-in-the-middle attack, the attacker is assumed to have the ability to view the entire message exchange and remove and replace packets. Such an attacker should have no way to associate a vote with a voter, and should be unable to interfere with the voter casting the ballot of their choice. We assume that the attacker is unwilling to be detected, and so cannot simply remove all messages to keep the voter from voting.

In a DoS attack, an attacker, who might be willing to risk detection, is trying to affect the conduct or the outcome of the election by effectively disabling one or more of the computers involved in the communication. Since a voting client computer is used for a limited time and may have dynamically assigned IP addresses, it makes a poor target for DoS attacks. We therefore focus on protecting the voting servers.

C. Assumptions

A secure channel, such as in-person registration or postal mail, allows the configuration of each voting client, and similarly for all the voting servers. For example, the voting clients must know the correct public keys, IP addresses, and port numbers of the servers. Each client must also obtain as many 16-byte random shared secrets as there are BASs, one secret shared with each BAS. As part of this configuration, all parties agree on the size of each field and the details of the protocol used, so protocol headers do not provide information about version number, field sizes, etc.

Each voting client must be able to produce a certain number of bytes of truly random data for use as keys and blinding factors. RFC 1750 [15] describes practical means of obtaining such truly random data, also available on modern operating systems, e.g. using `/dev/random` on Linux.

II. PROTOCOL OVERVIEW

The two sets of exchanges are shown in Figure 1. First the voting client sends the blinded ballot to the BASs and gets back a blinded signature. Next the voting client sends a single message to each VTS to record the vote. Each VTS confirms that the message was received.

A new ballot sent by a voter can be validated by any BAS with the database of legitimate voters. Likewise, a voting client

can verify that a signature from a BAS or an acknowledgement from a VTS is valid. A VTS can check that a signature for a ballot was produced by a given BAS.

The message to a VTS is encrypted using AES256, which prevents an attacker from recognizing a voter's choices. The same packet carries the AES256 key itself, encrypted using the VTS's RSA768 public key, so only that VTS can decrypt the signed ballot.

As long as the VTS does not record IP addresses, the voter remains anonymous. Alternately, if a corrupt VTS records IP addresses, anonymity is preserved as long as there is no way to associate the IP address with a voter.

The messages are sent using UDP at a relatively slow constant rate. If there are multiple signing servers, the client only needs signatures from a majority of these. As a consequence, loss of some packets or a minority of servers need not result in retransmissions. Likewise for VTSs, acknowledgement from any one VTS is sufficient, though two or more VTSs are preferable, with each serving as a backup to the other(s).

The remainder of this paper describes in detail the protocol and format of each message, then considers the protocol's resistance to common DoS attacks.

III. Protocol Details

For this protocol, a ballot is a sequence of arbitrary bits to be communicated to a VTS after the signing server has confirmed the voter's eligibility to vote by blindly signing the ballot. For this version of the protocol, the maximum ballot size is 223 bytes, for reasons explained below.

A. Signing the blinded ballot

The sequence of bits representing the vote are the basic ballot b. The ballot to sign, bts, is formed by a single byte of zero, 32 bytes of ballot ID, and the basic ballot b, for a total of 256 bytes, 2048 bits. The length of b is therefore 223 bytes. The bts must be 2048 bits since RSA2048 is used (in this version of DVVSB) to sign the ballot. The first byte is zero to ensure that the bts is less than the public key n of the signing server. BAS public keys must be selected so the first byte of n is nonzero, so that $n > bts$.

If the ballot is less that 223 bytes, the ballot is encoded as a sequence of three fields: a one-byte length field, the ballot data itself, and sufficient padding to make 223 bytes. The padding becomes part of the ballot, and should:

- not reveal any information about the voter
- not allow a known-plaintext attack on the ballot, and
- be computable given only the ballot ID and ballot data

For this version of DVVSB we have chosen to use as padding the last 31 bytes of the ballot ID, repeated and then truncated as needed.

The voter and the voting client together choose the randomly selected ballot ID. This ID uniquely identifies a valid, signed ballot, but not a voter. Multiple ballots with the same ballot ID and the same voter content can be assumed to be duplicates and subsequent submissions are ignored by the VTSs. Since 32 bytes is 256 bits, the chance of two voters randomly selecting

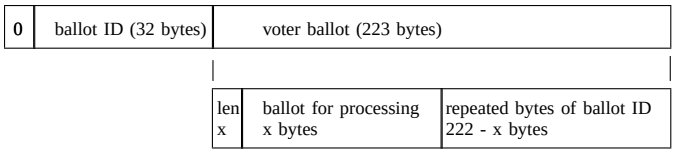

Fig. 2. Format of voter ballot to sign.

the same ballot ID when there are N voters is approximately $1 - e^{\frac{-(n(n-1))}{2^{257}}}$. For one million voters, the probability is approximately 2^{-217}. Holding 10,000 such elections a year, collision would be very unlikely in the lifetime of the universe.

To see that true randomness is needed, consider a pseudo-random number generator seeded with the seconds value of a clock during a 10 hour voting period, $36,000$ seconds. Among the same million voters, collisions are inevitable.

The voting client blinds bts by computing $bts^* = bts \times k^e \bmod n$, where k is a random blinding factor, and e and n are the BAS's public key (we follow the convention that $e = 2^{16} + 1$).

As long as k is truly random, there is no way for an attacker that has access to bts^* to identify bts. To see this, consider that for any potential ballot bts', one can always find a factor k' such that $k'^e = bts'/bts^*$. In other words, any hypothesized ballot can equally easily be obtained from the blinded ballot, and possession of the blinded ballot gives no information about the cleartext ballot. Our implementation obtains the k-inverse from `/dev/random`, and inverts it modulo n to obtain k.

To allow each BAS to quickly discard invalid ballots, the ballot submitted to the BAS also includes a SHA-512 HMAC computed over the blinded ballot and the shared secret that should be known only to the BAS and the voting client. The size of the shared secret is not defined by this protocol, but our current implementation uses 16-byte shared secrets. This shared secret is associated with a voter ID which must be unique to each voter but need not be kept secret, since it cannot be associated with a readable ballot. In this version of the protocol, the voter ID is 8 bytes. The voter IDs can be assigned sequentially, and most of these bytes may well be zero. Note that voter IDs are different from the ballot IDs used to identify ballots, are assigned at registration time, and are not secret.

The message from the client to the BASs then consists of 256 bytes of the ballot-to-sign (bts) blinded using the factor k, followed by the 8 bytes of voter ID, and the 64 bytes of the SHA512 code computed over the blinded ballot, the voter ID, and the secret shared between the BAS and the voting client. The voter ID lets the BAS quickly look up the shared secret, and the SHA512 HMAC lets the BAS quickly determine whether the sender is in possession of the correct shared secret.

Each BAS can use the same voter ID to identify the same voter, but must use a different random shared secret to minimize the consequences of a BAS being subverted.

A BAS receiving a 328-byte UDP packet first looks up the voter ID. If the voter ID is not valid, the packet is discarded. If the voter ID is valid, the BAS checks its storage to see if

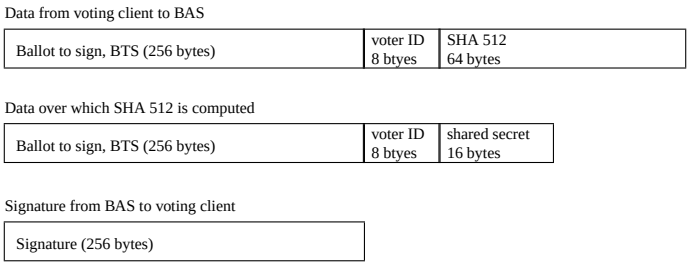

Fig. 3. Format of messages exchanged between voting client and BAS.

this voter has voted before. If so, either:

- the packet is the different and can be discarded, or
- the packet is the same as a previously submitted valid vote, and the stored answer is sent back.

These operations can be made very fast, so a BAS can quickly respond to packets with voter IDs it has seen before.

If the voter ID is valid but the voter has not voted before, the SHA512 computation is done over the packet, the voter ID, and the shared secret associated with this voter ID. If this computation does not match the 64-byte HMAC in the packet, again the packet is discarded.

Otherwise the packet is accepted. The BAS then records the packet and signs the blinded ballot part of the packet, bts^*, by raising it to the power of the BAS's secret key d modulo the public key n. This is sent back as a 256-byte UDP packet.

The voting client unblinds each 256-byte reply from a BAS by multiplying it by $k^{-1} \bmod n$, using the k^{-1} and n corresponding to the IP and port number of the BAS that replied. The client verifies that $\mathrm{bts} = s^e \bmod n$, showing that the signature is correct for the original ballot. If the signature sent by the BAS is not valid, the client simply ignores it, retransmitting its request(s) when the original timeout expires.

The voting client must gather signatures from at least a majority of the BASs. With one BAS, one signature is sufficient. If protection is desired against failure or collusion of up to m BASs, then there must be at least $3m + 1$ BASs, and at least $N_{sig} = 2m + 1$ signatures are required. With four servers, three signatures are required, and $m = 1$, so one corrupt server cannot affect the outcome of the election.

While this may recall Threshold Security [16], the public keys of the BASs are sufficient to verify a ballot, and no secret keys are needed.

The voting client retransmits requests to each BASs until it has received signatures from the required number of BASs. Assuming that sufficient bandwidth and computational capacity for the election has been provisioned, and to add some protection against denial of service attacks, the requests are retransmitted at a constant rate to each BAS that has not yet responded.

B. Submitting the ballot to be tallied

Once a client has obtained enough valid signatures, it sends a packet to the VTSs. This packet contains of the 256-byte ballot-to-sign (bts) followed by the valid signature records. Each signature record includes the last 8 bytes of the public

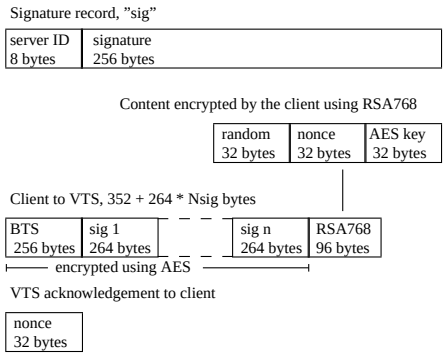

Fig. 4. Format of messages exchanged between voting client and VTS.

key of the signing server, followed by the 256-byte signature itself, for a total of 264 total bytes per signature record. The last 8 bytes of the public key of the signing server are used because:

- they are known to each client and VTS, and
- they are likely to be unique, and the voting authority can easily ensure that they are unique

Other 64-bit numbers with these properties could be used.

The ballot and the signatures are encrypted using AES 256 using a randomly generated 256-bit key $aesk$, different for each VTS. Counter mode is used with the initial counter value being the 256-bit number 2.

Next, a 32-byte random nonce and the 32-byte AES key aesk are concatenated, then padded with on the left (after a 0 byte at the beginning) with random bytes sufficient to give a 768-bit number. This 768-bit number is RSA encrypted for each VTS using that server's public key n, so $a = (0.rand31.nonce.aesk)^e \bmod n$. This 768-bit value a is appended to the ballot that was encrypted using AES, and sent to the corresponding VTS. The total size is $352 + 264 * N_{sig}$ bytes.

Each VTS listens for packets of size $352+264\times N_{sig}$ bytes. For packets of acceptable size, the VTS must:

1) use its RSA-768 private key to decrypt the AES-256 key and the nonce;
2) use the AES-256 key to decrypt the ballot and the signatures;
3) check to see if the same ballot has already been received, and if so, resend the prior reply, otherwise
4) verify that at least N_{sig} signatures match the ballot

If the last step fails, the packet is discarded. If these steps succeed, this is a newly received valid ballot. It is saved, and the 32-byte nonce is returned to the voting client as an acknowledgement.

Once the voting client receives the nonce from at least one of the VTSs to which it sent its signed ballot, it can assume that the vote was recorded successfully. Requiring at least a acknowledgements protects against failures, or other causes of data loss, in up to $a - 1$ VTSs.

The ballot can also be sent to trusted intermediaries, who can submit it indirectly, and return the nonce to the voter as evidence that the VTS has decoded the ballot. The voting client

may also retain the signed ballot for later resubmission. Since the signed ballot is entirely self-validating, it can only have been signed by someone with access to the BASs' private keys.

In summary, only legitimate voters may vote, and at most once. The ballot is valid as long as no more than m BAS keys have been revealed, and may be resubmitted if necessary.

IV. Resistance to Man-in-the-middle Attacks

For a successful man-in-the-middle attacks, an attacker with access to all the packets must be able to read or modify a voter's vote.

Blinding keeps an attacker from reading the messages to or from the BAS. Encryption, using AES for the contents and RSA 768 for the AES key, keeps secret messages to the VTS. The nonce returned by the VTS carries no information of any use to an attacker.

The BAS can confirm that a message comes from a client who has the shared secret. The client can verify that the signature comes from a BAS rather than from an attacker. The VTS can verify that the signatures match the ballot submitted by the client. The nonce returned by the VTS assures the client that the ballot was decrypted by the VTS.

We conclude that the DVVSB protocol prevents man-in-the-middle attacks.

V. Resistance to Denial of Service Attacks

Any attack that completely prevents packet exchange cannot be frustrated with an end-to-end protocol. Instead, we assume the network still delivers some fraction of the packets, as happens if routers drop packets as a result of a DoS attack.

Assume packets from a voting client are delivered to a server with probability p, and the response packets from the server are delivered with probability p'. $p = 1 - L$, where L is the traditional packet loss probability.

Many techniques can improve p and p', for example TVA [17]. We assume that all possible protective measures have been taken, and that p and p' are the resulting packet loss rates. Further, we assume that packet losses are independent and identically distributed.

In the simplest case of one BAS and one VTS, the probability that one exchange is successful is pp'. The probability that the two exchanges needed to vote are successful, is $(pp')^2$. For example, if $p = p' = 0.5$, the probability that voting completes without retransmission is $0.5^4 = 6.25\%$.

With retransmission, an average of pp' retransmissions are needed before a signature or ack is successfully received. Again, for $p = p' = 0.5$, an average of 4 transmissions to each server obtains a reply.

This compares favorably with the behavior of many conventional protocols. For example, using UDP over IPSec [18] requires at least 4 packets just to set up an encrypted session. TCP likewise also requires at least a three-way handshake with each server followed by at least two additional packets. The resulting probability of communication with one server, without retransmission, is $(pp')^2$, which is less than the probability of success pp' for one DVVSB data exchange.

When there is a DoS attack TCP reduces its sending rate, assuming that its traffic is the cause of the dropped packets. This makes it harder to accomplish the communication task, and easier to cause DoS. Sending at a fixed, relatively slow rate avoids this problem. Our implementation sends one packet to each server every 20 seconds.

Retransmission continues until the required number of signatures or nonces has been received. Faster or slightly slower fixed rates of retransmission would also be effective. The election traffic does not cause congestion as long as the network is provisioned to handle the expected peak traffic.

A. Specific DoS attacks

We consider three scenarios for DoS Attacks:

- **Random:** the attacker sends random data
- **Replay:** the attacker resubmits valid ballots
- **Fake:** the attacker submits fake ballots, data as close to a ballot as can be generated without access to the secret keys

Random packets being sent will usually be of a size different from valid data, and can easily be discarded by voting clients and servers alike.

The system is designed to handle **Replay**ed packets as retransmissions. As described in Section III-A, BASs can quickly re-send saved responses, and VTSs can as well. For example, our BAS implementation requires less than 200μs to verify the SHA-512 hash and to retrieve a previously saved signature. This allows the BAS to handle over 5,000 such 328-byte packets per second, or 13Mb/s of DoS **Replay** traffic. Voting clients will also discard duplicate messages.

This leaves the case of **Fake** data being sent by an attacker. Such packets have the same number of bytes as legitimate packets, so the servers and clients must verify them before being able to discard them.

There are four packet types to consider: blinded ballots sent to BASs, signatures sent back to voting clients, encrypted signed ballots sent to VTSs, and nonces sent back by VTSs to voting clients to acknowledge receipt.

If the network supports ingress filtering, the attackers cannot spoof the source IP address, and it is very hard to do a denial of service attack on the clients. Even without ingress source filtering, verifying a nonce is trivial and fast, and verifying returned signatures could be done at a rate of over 2Mb/s. Attacking any significant number of voting clients with this much traffic should be challenging, so in what follows, we focus on DoS **Fake** attacks on the voting servers.

1) DoS attacks on BAS: A BAS must verify each blinded ballot by first checking the voter ID. If the the corresponding voter has not voted yet, the SHA-256 HMAC is computed using the secret shared by the voter and the BAS. Our implementation reported a time of 71μs for one such computation, on a 2.4GHz Intel Core 2 quad-core CPU. Such a BAS should be able to discard well over $10,000$ invalid packets per second, or well over 30Mb/s of DoS **Fake** traffic.

The same BAS is capable of signing at least 100 valid ballots per second. If there is a need to sign ballots more

quickly, the computation can be distributed among different processors, each signing a separate ballot. This should scale linearly, as long as one processor is dedicated to distributing ballots to be signed by the other processors.

We also did a very simple DoS test on an isolated network of four relatively slow PCs (730MHz) connected through a 100Mb/s Ethernet switch. On this network, a throughput test using `ttcp` showed the receiver receiving 89Mb/s. On this network, the attacker sent *Fake* packets, with a valid user ID, from one host to the BAS at an average rate of 78Mb/s. The client was run 10 times, and had to retransmit an average of 7.4 times before receiving a signature. In the ten trials, the minimum number of transmissions was 1, the maximum 16, though in a separate test 24 transmissions were observed. In this simple test, the client retransmitted every second, so the response was relatively quick. With multiple BASs, all of them can be tried in parallel, so the expected time to collect multiple signatures is not significantly greater than for one signature.

2) DoS attacks on VTS: In addition to verifying the signatures, the VTS must decrypt a packet encrypted using RSA-768, and a possibly sizable packet encrypted using AES-256. In our implementation, it takes 1.5ms for the RSA decryption on the fast server (about 3ms on the 730MHz processors), about 0.5ms for the AES-256 decryption, and about 3.7ms to verify 3 signatures. Assuming a packet is discarded after failing the first signature verification, a VTS running on server-grade hardware should take about 3.2ms to recognize that a packet is not valid, and so be able to discard at least 300 invalid packets per second. If 3 signatures are needed, each valid packet has $1,144$ bytes, and the server can discard about 2.7Mb/s of DoS traffic. This performance is not sufficient to thwart a determined attacker.

In our simple DoS **Fake** test, each packet had one signature and so 616 bytes, and we sent about 1850 packets per second to the VTS, giving about 9Mb/s of DoS traffic. Ten trials averaged 10.3 client transmissions to get a response, with a minimum of 3 and a maximum of 18 transmissions.

It should be straightforward to improve VTS speed dramatically by taking advantage of the parallelism inherent in the problem. Each packet can be verified independently of every other packet, perhaps after a quick check in a Bloom filter or hash table to verify that it has not been received before. So with 128 processors, for example, it should be possible to handle over 300Mb/s worth of incoming 3-signature packets.

In addition, a voting client only needs a response from a single VTS. Many independent VTSs can be provided, and then their data merged at the end of the election, perhaps using removable storage. It is harder for an attacker to target many VTSs.

Our VTS can accept about 250 valid votes per second if 3 signatures are required. Again, this problem should scale linearly with the number of processors, so if higher performance is required, more machines can be used.

VI. Conclusions and Ongoing Work

We have introduced the basic design of DVVSB and explained why the networking protocol is resistant to man-in-the-middle attacks and some forms of DoS attacks.

We expect to continue implementing our prototype system, and use it to further evaluate the design, at our lab and on a larger scale such as at the DETER lab.

We are hoping to perform elections with the prototype in several local elections, including student board elections, neighborhood board elections, and educational voting projects in high schools.

We also hope to work with experts in human-machine interfaces to design a system that is not only secure and user-verifiable, but also usable and useful.

References

[1] E. Biagioni, Y. Dong, W. W. Peterson, and K. Sugihara, "Practical distributed voter-verifiable secret ballot system," in *Symposium on Applied Computing*, Honolulu, Mar. 2009.

[2] D. L. Dill. Openness and security. [Online]. Available: http://vote.nist.gov/speeches/2 - Security Panel/dill-nist.pdf

[3] M. Shamos, "Paper versus electronic records," Carnegie Mellon University, Tech. Rep., Apr. 2004. [Online]. Available: http://vote.nist.gov/threats/papers/paper_v_electronic_records.pdf

[4] Minnesota Secretary of State. (2008, December 12,) US senate recount data. [Online]. Available: http://electionresults.sos.state.mn.us/SenateRecount.asp

[5] Threat analyses & papers. Most interesting are the VVPAT and paper trail manipulation papers. [Online]. Available: http://vote.nist.gov/threats/papers.htm

[6] [Online]. Available: http://www.punchscan.org

[7] D. Chaum, A. Essex, R. T. Carback III, J. Clark, S. Popoveniuc, A. T. Sherman, , and P. Vora, "Scantegrity: End-to-end voter verifiable optical-scan voting," *IEEE Security & Privacy*, May/June 2008.

[8] R. L. Rivest, "The ThreeBallot voting system," MIT, Tech. Rep., Oct 2006. [Online]. Available: http://people.csail.mit.edu/rivest/Rivest-TheThreeBallotVotingSystem.pdf

[9] K. Fisher, R. Carback, and A. T. Sherman, "Punchscan: Introduction and systemm definition of a high-integrity election system," University of Maryland, Baltimore County, Tech. Rep., May 2006. [Online]. Available: http://www.punchscan.org/papers/fisher_punchscan_wote2006.pdf

[10] A. Fujioka, T. Okamoto, and K. Ohta, "A practical secret voting scheme for large scale elections," in *Advances in Cryptology, AUSCRYPT '92*, ser. Lecture Notes in Computer Science, J. Seberry and Y. Zheng, Eds., vol. 718. Berlin: Springer-Verlag, 1993.

[11] L. F. Cranor and R. K. Cytron, "Sensus: A security-conscious electronic polling system for the internet," in *Proceedings of the Hawai'i International Conference on Systems Sciences*, Wailea, Hawai'i, USA, January 7-10, 1997. [Online]. Available: http://lorrie.cranor.org/pubs/hicss/hicss.html

[12] D. Chaum, "Blind signatures for untraceable payments," *Advances in Cryptology - Crypto '82*, pp. 199–203, 1983.

[13] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, p. 120126, 1978.

[14] *Specification for the ADVANCED ENCRYPTION STANDARD (AES)*, National Institute for Standards and Technology (NIST) Std. 197, Nov. 2001.

[15] D. Eastlake, 3rd, S. Crocker, and J. Schiller, *Randomness Recommendations for Security*, IETF RFC 1750, Dec. 1994. [Online]. Available: http://www.apps.ietf.org/rfc/rfc1750.html

[16] H. Vu, N. Mittal, and S. Venaktesan, "THIS: THreshold security for information aggregation in sensor networks," in *International Conference on Information Technology (ITNG'07)*, 2007.

[17] X. Yang, D. Wetherall, and T. Anderson, "A DoS-limiting network architecture," *SIGCOMM Comput. Commun. Rev*, vol. 35, no. 4, pp. 241–252, 2005.

[18] VPNC. VPN protocols. A survey of VPN and IPSec references. [Online]. Available: http://www.vpnc.org/vpn-standards.html