# Hover: Trustworthy Elections with Hash-Only Verification

**Aleksander Essex** | Children's Hospital of Eastern Ontario Research Institute
**Urs Hengartner** | University of Waterloo, Canada

**Hover (Hash-Only Verification), an end-to-end (E2E) verifiable voting system with distributed trust, uses only a collision-resistant hash function for verification. Such verification could make E2E elections more accessible to people without a strong cryptography background.**

Recall an old engineering adage: "Fast, good, cheap—pick any two." It's a testament to the trade-offs inherent in any design project. Over the past 30 years of research in cryptographically end-to-end (E2E) verifiable elections, the trade-offs have been primarily among cryptographic verifiability, distribution of trust, a usable voting interface, and conceptually simple verification. Cryptographers have tended to focus on the former two, whereas election officials and voters have championed the latter two. In our view, compromise is necessary to advance the cause of trustworthy elections.

To that end, we propose Hover (*H*ash-*O*nly *Ver*ification), a novel combination of several recent ideas in the field of trustworthy voting. At Hover's core is the Scantegrity optical-scan system.[1,2] We also borrow aspects from the Eperio system to facilitate verification of the noncryptographic parts in familiar software applications such as desktop spreadsheets.[3] In addition, we use oblivious printing[4] and related techniques[5] to distribute trust among multiple election trustees without increasing the election audit's technical requirements. We believe that Hover can help educate less technical audiences about the merits of cryptographically verifiable elections.

As a consolidation of research ideas, Hover doesn't claim great technical innovation. Instead, it aims to strike a delicate balance of technologies to innovate at a social level. Although this pursuit is ongoing, we hope Hover can help bring cryptographers and the general public closer together regarding how to run trustworthy elections.

## Voters vs. Cryptographers

Designing cryptographically E2E verifiable elections is both a technical and social challenge. On a technical level, cryptography can make elections verifiable in a way that fundamentally exceeds conventional approaches. It offers each voter a means to check that his or her ballot was counted correctly, without revealing it to anyone. Such elections' trustworthiness is the product of statements proven directly about the election results, not of individual voting machines or their software.

However, on a social level, many people (in what cryptographers call the "real world") continue to wonder how a cryptographic proof can ever truly be convincing to average voters. Consider a recent German constitutional court decision on election verification: "Each citizen must be able to comprehend and verify the central

 Copublished by the IEEE Computer and Reliability Societies 

steps in the election's reliably and without any special prior technical knowledge."[6] A recent legal analysis by Denise Demeril and her colleagues is dubious as to E2E verifiable systems' future under such a ruling.[7]

Still, we believe cryptographic verification represents the future of elections. The 2009[8] and 2011 cryptographically verifiable elections in Takoma Park, Maryland, are a testament to progress in that regard. The challenge is to make the ideas behind E2E verifiable elections more conceptually accessible or risk marginalizing the technology. After all, if a proof is made in a forest—and no one is around to check it—is it complete and sound?

## Popular Proofs

Researchers often approach E2E verification system design as a trade-off. At one end is unconditional integrity. Under this model, election trustees can't reliably produce a false (but valid-looking) proof, even with access to sufficient computational resources. However, someone with sufficient computational resources could exploit this proof to break ballot secrecy.

At the other end is everlasting privacy. Under this model, a proof can never be used to recover the association between vote and voter. However, given access to sufficient computational resources, election trustees could potentially publish a false (but valid-looking) proof. Tal Moran and Moni Naor note everlasting privacy's applicability to the voting problem: "While integrity that depends on computational assumptions only requires the assumptions to hold during the election, privacy that depends on computational assumptions requires them to hold forever."[9] Unfortunately, the cryptographic primitives typically associated with information-theoretic privacy exceed the knowledge base of average voters.[10]

To create an E2E verifiable election that's conceptually simple yet offers a degree of cryptographic security, we designed a *popular proof*—one that's convincing (to some degree) on both a social and technical level. So, in designing E2E verifiable voting schemes, our modus operandi is as follows. Each citizen must be able to comprehend and verify the central steps in the election reliably with minimal special prior technical knowledge. If special prior technical knowledge is an unavoidable consequence of a cryptographically verifiable election's security requirements, it should be biased more toward the creation of a proof than its verification.

## E2E with a Dash of Hash

Why base our E2E proof on a cryptographic hash function? We consider two perspectives:

- a nonexpert technical audience with a basic cryptography background and a potential interest in election auditing and
- a general audience with no cryptography background but with an interest in learning about its underlying principles.

For the nonexpert technical audience, hashing is one of the most commonly encountered cryptographic primitives. Anyone who has downloaded open source software will likely have encountered a hash. In addition, hashing is one of the few primitives commonly found in the standard libraries of many programming languages. A file hash can even be computed directly from a Unix-based command line (for example, Linux and Mac OS), and numerous free websites and utilities exist for users of other operating systems. By comparison, many cryptographic primitives favored in the literature (for example, homomorphic encryption and zero-knowledge proofs) don't have widely available implementations, often requiring that they be implemented from scratch.

To the general audience, hashing provides an intuitive parallel with physical fingerprints. Such a metaphor seems potentially useful, given prime-time crime dramas' popularity. For example, such a show's viewers might understand that police couldn't easily identify a fingerprint that wasn't already in their criminal database (preimage resistance). They might also understand that finding two people with matching fingerprints is unlikely (collision resistance) and that finding another person whose fingerprints match their own would be even more unlikely (second preimage resistance).

Although this analysis is imperfect, we can reasonably expect to face fewer obstacles connecting a new concept to an existing, well-formed mental model. By comparison, many primitives favored in the literature aren't nearly as strongly connected to an existing model. For example, semantically secure additively homomorphic public-key encryption would require conveying three novel concepts instead of one: randomized encryption, adding "under the covers," and asymmetric keys—none of which has a strong physical analogy.

### A Special-Purpose Hash-Based Commitment

We employ a hash function (under specific assumptions) as a computationally hiding and computationally binding commitment scheme. Cryptographic commitments are at the core of several trustworthy voting protocols, including Scantegrity and Eperio. Several commitment schemes exist in the literature, such as Torben Pedersen's unconditionally hiding commitment scheme.[9] In this application, we're interested in using hashing for its relative technical and conceptual simplicity.

Briefly, a cryptographic commitment scheme $\mathtt{Comm}(m, r)$ takes message $m$ and randomness $r$ and

produces commitment $c$. $\texttt{Open}(c, m, r)$ takes $c$, $m$, and $r$ and returns `accept` if and only if $\texttt{Comm}(m, r) = c$. A commitment is binding if it's hard to find any $\{m, m', r, r'\}$ where $m \neq m'$ such that $\texttt{Open}(\texttt{Comm}(m,r), m', r')$ returns `accept`. A commitment is hiding if finding any $\{m, r\}$ given $c$ such that $\texttt{Open}(c, m, r)$ returns `accept` is difficult. For a stronger hiding requirement, extracting partial information about $m$ given $c$ should also be difficult.

However, given $m$ and a collision-resistant hash function $H$, $H(m)$ isn't a commitment to $m$ in general. For one thing, hashing a message isn't computationally hiding when $m$ isn't chosen from a sufficiently large space. For example, committing to a single bit $b \in \{0, 1\}$ by posting $c = H(b)$ is trivially opened by checking which of $H(0)$ and $H(1)$ produce $c$. However, under certain circumstances, such as when the message is a sufficiently large random factor, a hash could function as part of an application-specific commitment. Although a commitment function based on a collision-resistant hash function can generally be considered a "false solution," we believe it might be suitable for our needs.

Given $H : \{0,1\}^* \rightarrow \{0,1\}^\ell$ (where $\ell$ is the length of the hash function's output in bits) and an efficient description of permutation space $\Pi$, for $|\Pi| \geq 2^\ell$, we define an application-specific commitment function as follows. $\texttt{Comm}(\pi)$ takes permutation $\pi \in_R \Pi$ and produces $c = H(\pi)$. $\texttt{Open}(c, \pi')$ takes $c$ and an asserted permutation $\pi' \in \Pi$ and returns `accept` if and only if $H(\pi') = c$.

For $\pi, \pi' \in \Pi$, `Comm` is binding as long as an adversary can't find

- a collision—that is, $H(\pi) = H(\pi')$ for any $\pi \neq \pi'$—or
- a second preimage—that is, given $\pi$, a $\pi' \neq \pi$ such that $H(\pi) = H(\pi')$.

We assume that finding valid collisions as part of the collision-resistant properties of $H$ and finding valid second preimages as part of the one-way properties of $H$ are computationally infeasible. `Comm` is hiding as long as an adversary can't exhaustively search the message space or invert the hash function—that is, given $c$, finding a $\pi \in \Pi$, such that $H(\pi) = c$, should be difficult. We assume $\pi$ is chosen uniformly at random from $\Pi$ and that $|\Pi|$ is sufficiently large that exhaustively searching the message space is computationally infeasible. We also assume that inverting $H$ as part of the one-way properties of $H$ is computationally infeasible.

An important question is whether collision resistance implies one-wayness. Mihir Bellare and Phillip Rogaway showed that an adversary with the ability to attack the one-wayness of $H : D \rightarrow R$ has a negligible advantage over an adversary with the ability to attack the collision resistance of $H$ when $|R|/|D|$ is also negligible.[11] For typical real-world elections, a collision-resistant hash function will imply one-wayness for our purposes.

In our application, the message we're committing to (that is, the hash image) is a random permutation of a list of the mark state (that is, marked or unmarked) of each optical-scan oval on each cast ballot. In an election involving $c$ candidates and $v$ voters, this list consists of $cv$ elements, with $(cv)!$ possible permutations. For example, for $c = 2$, $v = 50$, and a hash function with $\ell = 256$, we have $|D| = 100!$, $|R| = 2^{256}$, and $|R|/|D| \approx 2^{-269}$, which is *negligible*. A function $f$ is negligible if there is a positive integer $k$ such that $f(k) < 1/poly(k)$ for any positive polynomial $poly(\cdot)$.

### Small Information Leakages

As Shai Halevi and Silvio Micali warned, despite $H$ being collision resistant and one-way, there are no guarantees about the feasibility of computing partial information about $m$ given $H(m)$.[12] Their point was that to ensure the commitment is hiding, we must require additional properties of $H$. For this article, we exclude the threat of partial information leakage—for example, by modeling $H$ as a random function. This somewhat contradicts our real-world setting, so it's worth briefly outlining potential consequences when an adversary can extract a small number of bits from $m$ given $H(m)$.

Although we believe that the overall threat to voter privacy is low, proving this would be complex and would require specific assumptions about the leakage's distribution for a given hash function. Still, aspects of our system offer an inherent degree of fault tolerance. For example, a leakage of $g$ bits isn't sufficient to determine any single pair $\{x, \pi(x)\}$ as long as $\log_2((cv)!)/cv > g$. So, leaking a small number of bits (for example, one or two) would never be sufficient to compromise the secrecy of any ballot for most election sizes. Intersection attacks combining partial leakages across multiple proof instances remain possible. However, on the basis of previous (unpublished) Scantegrity analyses, we contend this threat is fairly minimal for small leakages.

## Basics and Election Setup

For simplicity, we describe the election correctness proof as being generated by a single trusted authority. A trusted authority can't create a false (but accepting) proof, but, as in Scantegrity, it does receive sufficient information to link individual voters to their voting intentions. Recent advances in secure document printing can be used to distribute trust among multiple authorities.[4]

A single election trustee $T$ initializes an election by establishing a public append-only bulletin board $BB$. $T$ defines $\Pi$, which consists of all possible permutations of $cv$ elements. $T$ posts $c$, $v$, an efficient description of $\Pi$, a description of $H$, a canonical list of candidate

names $N$, and an alphabet of $c$ valid or possible confirmation codes $D$ to $BB$. Proving the association between codes and candidates printed on the ballot is correctly reflected on the bulletin board requires this information to be made public. Some ballots are randomly selected for this "print audit." These ballots can't be voted on, so in practice, $v$ is several times larger than the actual number of registered voters. $T$ generates $v$ ballots, each containing $c$ unique tuples $\{s, d, n\}$ associated with a specific optical-scan oval on the ballot. This tuple consists of a ballot-specific serial number $s \in \{1, \ldots, v\}$, a code letter $d \in D$ assigned randomly without replacement, and the associated candidate name $n \in N$. The association between a given code $d$ and candidate name $n$ is independent and random across ballots. All $v$ ballot tuples form a master ballot table $B = \{B_s, B_d, B_n\}$. $B$ is given to a trusted printer who prints each of the $v$ ballots.

For $i = 1 \ldots p$ proof instances, $T$ chooses two random permutations $\rho_i, \sigma_i \in_R \Pi$ and computes shuffled candidate list $B_{n_i} = \rho_i \circ \sigma_i(B_n)$. $T$ posts $B_s$, $B_d$, each of the $p$ shuffled candidate lists $B_{n_i}$, and commitments to the associated random permutations, $H(\rho_i)$ and $H(\sigma_i)$, to $BB$.

A voter marks the optical-scan oval appearing beside the chosen candidate's name. The voter creates a receipt $r = \{s, \hat{d}\}$ by writing down the serial number $s$ appearing on the ballot and the code letter $\hat{d}$ appearing beside the marked oval. Figure 1 depicts a ballot, its receipt, and the associated bulletin board entry.

Because receipt creation is unsupervised, disputes might arise between a voter and $T$ over which code was marked. In its basic form, Hover uses the in-person dispute resolution from the original Scantegrity proposal.[13] This approach uses a noncryptographic cut-and-choose proof, which fits with our design goals but doesn't scale well and is cumbersome to administer. Scantegrity II uses invisible ink to offer an informational dispute-resolution procedure based on knowledge of a secret confirmation code. This procedure is conceptually more complicated and wouldn't be suitable for a distributed-trust setting. We recently expanded it to a distributed-trust setting,[5] but this comes at the conceptual cost of a zero-knowledge proof. Designing a dispute-resolution procedure that is scalable, uses an economy of cryptography, and yet is secure in a distributed-trust setting is a matter for future research.

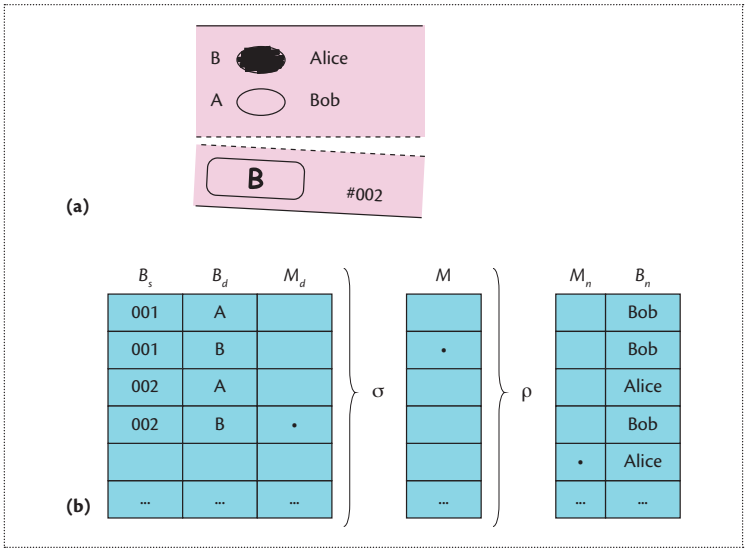


**Figure 1.** (a) A marked ballot and receipt. (b) A bulletin board showing receipt and candidate information including serial numbers $B_s$, confirmation codes $B_d$, and the mark state of the associated optical-scan ovals $M_d$. A secret shuffle dissociates a list of candidate names $B_n$ and the associated mark states $M_n$ from the corresponding entries in $B_s$, $B_d$, and $M_d$. The shuffle is expressed as the composition of two independent random permutations $\sigma$ and $\rho$—one may later be challenged by the public to be revealed as part of a cut-and-choose proof of correctness. Additional independent shufflings can increase soundness.

## Linkage Lists

Whereas Scantegrity publishes separate commitments to each pair $\{x, \pi(x)\}$, Hover commits to the full specification of random permutations $\sigma_i$ and $\rho_i$. Committing to an entire permutation not only lets us use a simpler commitment scheme but also results in substantially fewer cryptographic operations overall: $2p$ commitments for Scantegrity and $2pcv$ commitments for Hover.

An important part of a trustworthy election is the voter's ability to receive assurance that the bulletin board tables faithfully represent the code-candidate association appearing on the paper ballot. In a Scantegrity election, a voter (or some other designated third-party auditor) might challenge $T$ to prove the correct printing of any ballot. For each element in $B_s$ and $B_d$ associated with serial number $\hat{s}$ of the challenged ballot, $T$ discloses each associated element in each of the $p$ candidate lists $B_{c_i}$ by opening the commitments to the associated I/O pairs in $\rho_i$ and $\sigma_i$ ($2pc$ openings in total). Ballot $\hat{s}$ is then considered spoiled and can't be voted on; a poll worker marks the paper ballot accordingly, and a flag is placed beside the associated entries in the bulletin board.

Because Hover commits to permutations as a whole, it can't reveal individual I/O pairs directly. To address this, we use Eperio's *linkage list* construction.[3] Instead of proving code-candidate links by directly opening the affected commitments, $T$ merely asserts the links in a linkage list $LL$. These assertions' correctness can later be confirmed (with high probability) as part of the post-election audit. Although we feel this approach provides much simpler verification through full-permutation commitments, a major drawback is that printing faults are discovered only after the election.

An important question is whether our hash-based commitment securely composes by using a linkage list. Through the linkage list, we're intentionally disclosing partial information about $\rho_i$ and $\sigma_i$. This by itself isn't an issue: the number of unaudited ballots is always at least as large as the total number of voters. This means that the effective permutation space $\Pi_{ef} \subseteq \Pi$ will still be large despite partial disclosures owing to the print audit. However, a real-world cryptographic hash function might leak additional bits owing to this partial disclosure. Of course, if $H$ is a random function, adversaries can't use information revealed by the print audit to compute unknown bits.

## Election Challenges and Audit

After the election, $T$ uses the data collected by the optical scanners to create a mark state list $M_d$ of cast ballots. $M_d(i) \leftarrow 1$ if and only if the confirmation code $B_d(i)$ on ballot $B_s(i)$ was recorded as having been marked; $M_d(i) \leftarrow 0$ if it was unmarked. $M_d(i) \leftarrow \varnothing$ if the ballot was spoiled or print audited. $T$ posts $M_d$ to *BB*. Finally, for each proof instance $i = 1 \ldots p$, $T$ creates shuffled marks tables $M_i = \sigma_i(M_d)$ and $M_{c_i} = \rho_i(M_i)$ and posts these lists to *BB*. A shuffled candidate name list $B_{n_i}$ and the associated marks list $M_{n_i}$ are sufficient to compute an election tally. Anyone can check that each of the $p$ instances produces the same election result.

**Challenges.** For each proof instance $i = 1 \ldots p$, the public will collectively issue a challenge to $T$ to reveal either $\sigma_i$ or $\rho_i$. $T$ responds by opening the associated commitments. There are several ways to produce challenge bits; the essential property is that $T$ mustn't be able to predict the outcome with an advantage over a random guess.

One potentially suitable source of entropy for small-scale elections is the upcoming random beacon being implemented by the US National Institute of Standards and Technology (NIST).[14] The NIST beacon will draw on hardware-based entropy sources, offering the public digitally signed random bit strings in a persistent online database. Although the NIST beacon generally will be agnostic about what its outputs are being used for, it still represents a trusted component in this context, which might not be suitable for larger elections.

An alternative is randomness extracted from financial data—a method we used in the 2009 Takoma Park election. Jeremy Clark and Urs Hengartner analyzed this method and offered suggestions for securely combining the outputs of multiple independent sources.[15]

**Audit.** Voters can use their receipt $r = \{s, \hat{d}\}$ to check that, for each row $s \in B_s$, all the associated mark states in $M_d$ are unmarked, except for the one associated with their recorded confirmation code $\hat{d}$ in $B_d$. Anyone can check that the mark states were correctly propagated between the receipt and candidate tables by checking against the opened permutations—that is, by checking that $M_i = \sigma_i(M_d)$ or $M_{n_i} = \rho_i(M_i)$.

Auditors first verify the printing of challenged ballots by checking that the code-candidate associations indicated in the linkage list match those indicated on the corresponding paper ballot. Then, they check partial permutations in the linkage list against the opened permutations.

## Verification in a Spreadsheet

A common roadblock for many real-world implementations of cryptographic election verification is the software's relative complexity. Unlike conventional software, which can effectively function as a black box, election verification software must be developed in a functionally and conceptually transparent way. Consistent with our design goals, the election protocol should allow the verification software to be written with an economy of LOC. Owing to the nature of highly specialized cryptographic components necessary in many crypto voting protocols, the option to use existing software often doesn't exist.

Borrowing from Eperio's design goals,[3] we believe a verifier should be able to audit a cryptographic election using existing software as much as possible or, from the other perspective, be required to use as few LOC as possible. With Eperio, we presented a small audit tool written in 50 lines of Python.

As an alternative—one requiring no new LOC—we presented an example in which an election could be manually audited using a desktop spreadsheet. Although a manual audit would be tedious for large-scale elections, the widespread use of spreadsheets presents an interesting opportunity for explaining the election audit in familiar terms. In this way, the audit can be communicated as a series of basic spreadsheet operations, such as copying and pasting columns of data and simple commands such as sort and find.

To facilitate spreadsheet verification, $T$ encodes each ballot and mark list ($B_s$, $B_d$, and so on) as a comma-separated values (CSV) file. The random permutations $\rho_i$ and $\sigma_i$ are expressed as lists of shuffled integers $1 \ldots cv$, also encoded as CSV files. Hashes of permutation files are posted as commitments.

$T$ responds to the postelection challenges by posting the relevant permutation files. The verifier first checks that the asserted file is valid; that is, it's in the correct encoding and the file contains only a valid shuffle of the integers $1 \ldots cv$. The verifier then executes a command-line file-hashing utility and compares the result to the posted commitment. For example, if $H$ is SHA-256, the

verifier could run the `sha256sum` file-hashing utility. Optionally, these tasks can be automated using a simple shell script. Once satisfied that the commitments are correct, the verifier opens the file in a spreadsheet—for example, OpenOffice Calc.

The audit steps are followed mostly as we described earlier. To perform a permutation—for instance, to check whether $M_i = \sigma_i(M_d)$—the verifier copies the list of shuffled integers from the $\sigma_i$ worksheet and pastes them into the $M_d$ sheet. The verifier then sorts the sheet using the shuffled integers as the key. The verifier pastes $M_d$ (now shuffled) into the $M_i$ worksheet and tests the lists for equality. The linkage lists are checked similarly. Several built-in spreadsheet commands exist for efficiently performing such comparisons, and the audit can be automated by a macro.

## Distributing Trust

So far, we've described $T$ as a single trusted entity—that is, one who knows the associations between receipts and ballots (between voters and cleartext votes). Scantegrity extensively uses trusted entities and hardware: the optical scanners, ballot printers, and even poll workers gain access to receipt-and-vote combinations. However, because of the physical nature of the paper optical-scan ballot, distributing trust in this setting has proven challenging.

To overcome the issue of an untrusted ballot printer, we use oblivious printing.[4] Using a secure multiparty protocol and the invisible-ink printing techniques developed for Scantegrity, we propose a means by which several parties can generate a shared secret and print it in human- or machine-readable form without learning the result. Briefly, the secret is a combination of visual crypto shares successively overprinted in invisible ink. The recipient of an obliviously printed document can use a special pen to activate the ink, thereby revealing the message. This offers us a starting point for distributing trust in a voting setting.

Let $T$ be replaced by a collection of $t$ trustees $T_1 \ldots T_t$. First, the trustees run an oblivious-printing protocol to randomly generate and obliviously print the confirmation codes on each of the $v$ ballots. This protocol's output includes the obviously printed ballots and a vector of encryptions of each code-candidate association for each ballot. These encryptions are made using a semantically secure public-key encryption scheme for which the decryption key is distributed among the trustees.

For each proof instance $i = 1 \ldots p$, each of the $j = 1 \ldots t$ trustees separately generates and posts commitments to two random permutations $\rho_i^j, \sigma_i^j$. Next, the trustees compute $B_{n_i}$ using a reencryption mixnet, taking the encrypted candidate names as input and decrypting the shuffled result. The overall permutation on $B_{n_i}$ is thus $\rho_i^t \circ \rho_i^{t-1} \circ \ldots \circ \sigma_i^2 \circ \sigma_i^1$. The trustees similarly apply these permutations when populating the marks lists after the election. During the audit challenge, each trustee $T_j$ opens the commitment to the requested permutation, either $\sigma_i^t$ or $\rho_i^j$. The audit proceeds as we described earlier, except that the verifiers check a composition of the trustees' permutations—that is, checking either $M_i = \sigma_i^t \circ \ldots \circ \sigma_i^1(M_d)$ or $M_{n_i} = \rho_i^t \circ \ldots \circ \rho_i^1(M_i)$.

This approach lets Hover run with a distributed set of trustees and without a trusted printer. However, this approach doesn't offer fully distributed trust. The confirmation code associated with the chosen candidate is still revealed to the optical scanner (and potentially the poll workers) when a voter casts a ballot. We recently addressed this issue by defining two independent serial numbers: one for the ballot and one for the receipt.[5] We discussed the additional procedures necessary such that no single entity in the election, including the voter, ever sees both serial numbers. Finally, although the verifier must verify the openings of $t$ times as many commitments, at a conceptual level, the audit procedure is essentially the same as in the single-party case.

Regarding oblivious printing, two additional security requirements are necessary to prevent misbehaving printers from being able to link voters with votes.[5] The first requirement is a cut-and-choose-based proof made among the printers that each party followed the protocol honestly.[4] This audit would begin before the election and, assuming we want to avoid reliance on a physical chain of custody, continue during the election to ensure the ballot papers' provenance. The second requirement is that the printers must be prevented from examining marked ballots after the election.

In terms of cryptographic verifiability, Hover provides computationally sound election audits similar to those proposed in Scantegrity and Eperio. With regard to distribution of trust, oblivious printing lets voters cast an optical-scan paper ballot and construct a receipt without anyone else finding out for whom they voted.

In terms of a usable voting interface and conceptually simple verification, it's harder to ascertain how Hover fares without doing a usability study. A user study performed in Takoma Park suggested that voters and election officials found Scantegrity II ballots reasonably intuitive.[8] However, it's unclear how the changes introduced by oblivious printing would alter these perceptions; this is a question for future research. We propose a hash-based commitment, which we argue simplifies election verification at both conceptual and procedural levels.

Ultimately, we hope Hover will be a stepping stone toward the general public's greater acceptance and awareness of cryptographic election verification's merits. ■

## References

1. D. Chaum et al., "Scantegrity II: End-to-End Verifiability for Optical Scan Election Systems Using Invisible Ink Confirmation Codes," *Proc. 2008 Conf. Electronic Voting Technology* (EVT 08), Usenix Assoc., 2008, article 14; http://static.usenix.org/events/evt08/tech/full_papers/chaum/chaum.pdf.
2. D. Chaum et al., "Scantegrity II: End-to-End Verifiability by Voters of Optical Scan Elections through Confirmation Codes," *IEEE Trans. Information Forensics and Security*, vol. 4, no. 4, 2009, pp. 611–627.
3. A. Essex et al., "Eperio: Mitigating Technical Complexity in Cryptographic Election Verification," *Proc. 2010 Int'l Conf. Electronic Voting Technology/Workshop Trustworthy Elections* (EVT/WOTE 10), Usenix Assoc., 2010; www.usenix.org/event/evtwote10/tech/full_papers/Essex.pdf.
4. A. Essex and U. Hengartner, "Oblivious Printing of Secret Messages in a Multi-party Setting," to be published in *Proc. 16th Conf. Financial Cryptography and Data Security* (FC 12), 2012; http://fc12.ifca.ai/pre-proceedings/paper_58.pdf.
5. A. Essex, C. Henrich, and U. Hengartner, "Single Layer Optical-Scan Voting with Fully Distributed Trust," *Proc. 3rd Int'l Conf. E-voting and Identity* (VoteID 11), LNCS 7187, Springer, 2011.
6. *Verfahren über die Wahlprüfungsbeschwerden*, Judgment of 3 Mar. 2009, 2 BvC 3/07 and 2 BvC 4/07, Federal Constitutional Court of Germany (Bundesverfassungsgericht), 2009.
7. D. Demeril et al., "Feasibility Analysis of Prêt à Voter for German Federal Elections," *Proc. 3rd Int'l Conf. E-voting and Identity* (VoteID 11), LNCS 7187, Springer, 2011.
8. R.T. Carback et al., "Scantegrity II Municipal Election at Takoma Park: The First E2E Binding Governmental Election with Ballot Privacy," *Proc. 19th Usenix Conf. Security*, Usenix Assoc., 2010; http://static.usenix.org/events/sec10/tech/full_papers/Carback.pdf.
9. T. Moran and M. Naor, "Split-Ballot Voting: Everlasting Privacy with Distributed Trust," *Proc. 14th ACM Conf. Computer and Communications Security* (CCS 07), ACM, 2007, pp. 246–255.
10. T.P. Pedersen, "Non-interactive and Information-Theoretic Secure Verifiable Secret Sharing," *Advances in Cryptology—CRYPTO 91*, LNCS 576, Springer, 1991, pp. 129–140.
11. M. Bellare and P. Rogaway, "Introduction to Modern Cryptography," *CSE 207 Course Notes*, Dept. Computer Science and Eng., Univ. of California, San Diego, 2005, p. 207.
12. S. Halevi and S. Micali, "Practical and Provably-Secure Commitment Schemes from Collision-Free Hashing," *Advances in Cryptology—CRYPTO 96*, LNCS 1109, Springer, 1996, pp. 201–215.
13. D. Chaum et al., "Scantegrity: End-to-End Voter Verifiable Optical-Scan Voting," *IEEE Security & Privacy*, vol. 6, no. 3, 2008, pp. 40–46.
14. M.J. Fischer, M. Iorga, and R. Peralta, *A Public Randomness Service*, tech. report, US Nat'l Inst. Standards and Technology, 2011.
15. J. Clark and U. Hengartner, "On the Use of Financial Data as a Random Beacon," *Proc. 2010 Int'l Conf. Electronic Voting Technology/Workshop Trustworthy Elections* (EVT/WOTE 10), Usenix Assoc., 2010; www.usenix.org/event/evtwote10/tech/full_papers/Clark.pdf.





**Aleksander Essex** is a postdoctoral fellow at the Children's Hospital of Eastern Ontario Research Institute's Electronic Health Information Laboratory. His research interests include information security and cryptography, secure multiparty computation, and trustworthy voting and cryptographic election verification. He holds a postdoctoral fellowship from the Natural Sciences and Engineering Research Council of Canada and has a PhD in computer science from the University of Waterloo. He's a student member of IEEE. Contact him at aessex@ehealthinformation.ca.

**Urs Hengartner** is an associate professor in the David R. Cheriton School of Computer Science at the University of Waterloo, Canada. His research interests include information privacy; computer and network security; and security and privacy in emerging computing environments such as location-based services, geosocial networking, and e-voting. Hengartner has a PhD in computer science from Carnegie Mellon University. Contact him at uhengart@cs.uwaterloo.ca.