The 2024 Senate Selection Case Study on heain-core

A New Idea: Blind-Signature Tokens

The Voter's Device Blinds the Token Before the Roll Signs It, per RFC 9474, So the Roll and the Voting Machines Together Still Cannot Link Voters to Ballots

Jakkrit Boonma (จักรกฤษ บุญมา)

Independent Researcher · ORCID: 0009-0000-7165-3787 · jaggid.boonma@yahoo.co.th

Date: 30 October 2026

Implementation: heain-core 7ac7f80 (Go 1.26), core unchanged, HP Z840

Repository: github.com/JakkritB/heain-core (casestudies/senate/apps: blind.go, token.go, tokenkeys.go)

Abstract

Volume 1 [1] applied the HEAIN Framework to the existing Senate selection system and separated the identity lane from the vote lane with heain-core's mechanisms. But the token that grants the right to vote was still issued the simple way: the roll created the token itself and saw both the requester and the token at the moment of issue. If the roll kept that pair, it could link voters to ballots, every one. The secrecy of the ballot therefore rested on reviewing the app's code.

This volume proposes a new idea: the voter's device creates the token itself, blinds it with the roll's public key, and has the roll sign it after registration. The roll sees who asks but never sees the token; the voting machine sees the token but not who asked. It uses RSA blind signatures per RFC 9474 [3], variant RSABSSA-SHA384-PSS-Randomized, with keys per group and round, and publishes the key set as a signed statement on the public log, so the roll cannot use a per-person key to tag a voter.

On one machine (HP Z840) the results match every value of the RFC's test vectors. Putting together everything the rolls of all 6 districts saw (289 blinded requests) and everything every voting machine saw (288 tokens), no value appears in both. A roll that signed one voter's request with an unpublished key was caught by that voter's device. Rounds 0–5 and 2b of the case study, rerun on blind tokens, passed all 155 checks with no change to core. What the token still reveals is the occupational group, which at district level has about 2.5 voters on average, so anonymity is bounded by the group, the same scope as ballots sealed per group. All figures come from a simulation.

Keywords: *HEAIN Framework, heain-core, Senate Selection, Blind Signatures, RFC 9474, RSA-PSS, Unlinkability, Key Transparency, Tagging Attack, Anonymous Credentials, Electronic Voting*

1. Introduction

1.1 The problem Volume 1 left open

In Volume 1 [1], each round of the selection begins at the district roll. A candidate shows an ID card to an official; the roll checks eligibility and issues one token per person per round. The voter takes the token to a voting machine, which checks the roll's signature, accepts each token once, and seals the ballot at once. heain-core refuses to let the same value appear on both the identity lane and the vote lane (lanes.unlinkable), but core cannot see an app's memory. The roll creates the token's id itself; if it kept the id paired with the requester's name and then obtained the list of receipts, or the ballots opened for an inquiry, it could link every one.

This limit matters especially for the Senate case, because the allegation in the real event was organized voting from slips [1]. A system in which those with power over the roll can link voters to ballots would become a better tool for checking hired voters than publishing the counts, because a buyer would no longer need markers or published counts but would see each hired voter's ballot directly. Volume 1 therefore listed this among its limitations and proposed blind signatures as the way to close it.

1.2 The idea

A blind signature [2] lets a signer sign a message that has been blinded. It is like signing over an envelope with carbon paper inside: the signer knows whom they signed for but does not see the message in the envelope. When the requester opens the envelope, the signature sits on the real message and verifies against the signer's public key as usual, but the signer cannot link that signature to any signing session. It fits the token exactly: the roll checks eligibility and signs once per person per round, but does not know which token belongs to whom.

In RSA form, let m be the PSS-encoded message. The requester draws a random blinding value r and sends m' = m · r^e mod n. The signer returns s' = (m')^d mod n = m^d · r mod n (because r^(e·d) = r mod n), and the requester unblinds with s = s' · r^(−1) mod n = m^d mod n, an ordinary signature on m. Because r is fresh every time, the m' the signer sees reveals nothing about m. RFC 9474 [3] specifies the safe details of this scheme, including PSS encoding [4] and prepending a random value to the message.

1.3 Research questions

1. **RQ1** If the roll and the voting machines keep everything they see and put it together, can they link voters to ballots?
2. **RQ2** Can a dishonest roll tag some voters with a special key without being caught?
3. **RQ3** Does the existing system of Volume 1 (sealed ballots, counting under a grant, the lottery, the watch, the inquiry and the public log) work as before, and does core need changing?
4. **RQ4** What does the new token still reveal about the voter?

As in Volume 1, the candidates, votes and behaviour are all synthetic, and provinces and districts have made-up names. Voting by machine would require changing the law. This volume proposes an idea and simulation results; it does not claim to be ready for real use. The deployment is the author's hypothetical design, not a project of the Election Commission or any agency.

2. The Proposed Design

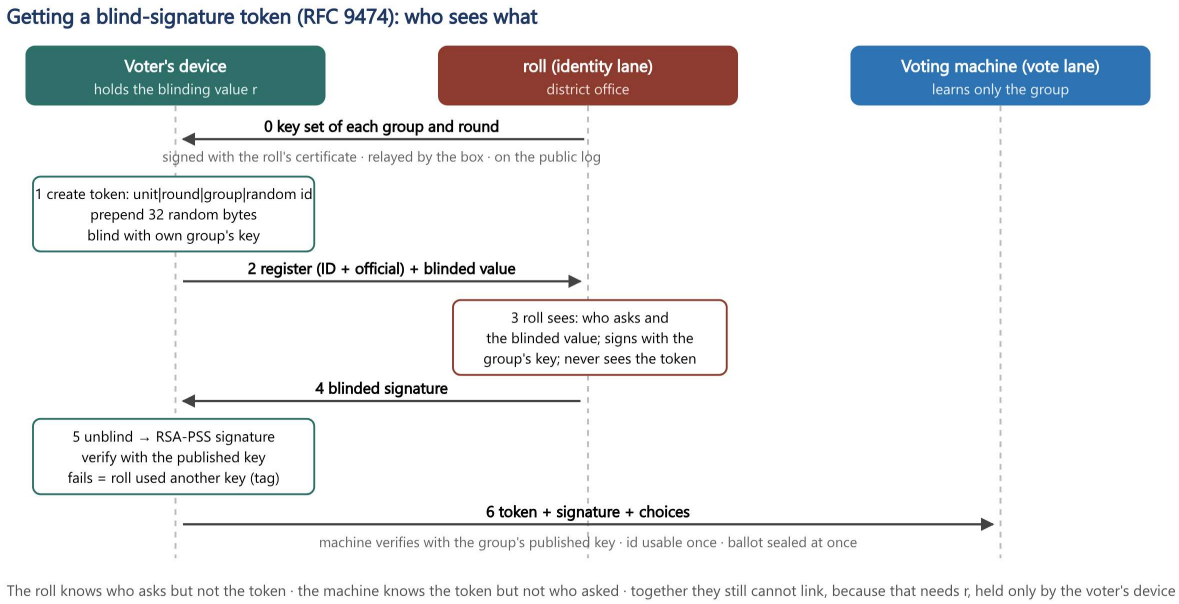


Figure 1 — Getting a blind-signature token, and what each party sees

2.1 Steps

1. **Key set:** the roll has one RSA-2048 key per occupational group per round (20 groups × 2 rounds) and signs the set of public keys with its app certificate. The set is relayed through the box and published on the public log before registration opens.
2. **The voter's device creates a token:** the message "senate-token/v1|unit|round|group|random id", with 32 random bytes prepended per the Randomized variant of RFC 9474, blinded with the key of the voter's group.
3. **Registration:** the voter shows an ID card to an official as in the existing system and submits the blinded value. The roll checks eligibility and that it has not already issued one this round, then signs with the key of that voter's group.
4. **Unblinding:** the device obtains an ordinary RSASSA-PSS (SHA-384) signature on the real message and verifies it against the published key. If it fails, the roll did not use the published key.

5. **Voting:** the machine reads the group from the token, picks that group's published key, verifies the signature, the unit, the round and the group's table, accepts the id once, then seals the ballot with core as in Volume 1.

2.2 Design decisions

Matter	Choice	Reason
Signature scheme	RSA blind signatures, RFC 9474, RSABSSA-SHA384-PSS-Randomized	An IETF standard with test vectors to check against; the final signature is RSA-PSS, which standard libraries verify
Who blinds	The voter's own device	Even if the roll and the voting machines collude they cannot link, because the blinding value is with neither (author's decision, 30 Oct 2026)
A key per group per round	20 × 2 keys per unit	The signer cannot see the token's content, so group and round are bound to the token through the key used
Where private keys live	The roll's register, sealed by core (identity lane)	Keys sit in the same scope as the identity data and are destroyed with it after the objection period
Publishing the key set	A signed statement relayed through the box once, and on the public log (C11)	Everyone in a unit gets the same set, so the roll cannot hand out per-person keys to tag voters
Voting order	Everyone collects a token first, then votes in shuffled order	Ballot order does not reveal registration order
Changes to core	None	Blinding and signing are the app's work; core does what it already does: sealing, lane separation, P5 and the public log

Table 1 — Main design decisions and their reasons

2.3 Who sees what

Party	Simple token (Volume 1)	Blind token
Roll	Requester + token id + signature	Requester + blinded value + blinded signature
Voting machine	Token + choices (before sealing)	Token + choices (before sealing)
Roll + machine together	Can link requester to ballot, every one	Cannot link; needs the blinding value held on the voter's device
Inquirer opening ballots under a grant	Ballots carry no voter, but a roll that kept ids could link them	Ballots carry no voter, and no one can link them
Outsiders (public log)	No token data	The key set of each unit and round

Table 2 — What each party sees with the simple token (Volume 1) compared with the blind token

The real change is in the third and fourth rows. In Volume 1, ballot secrecy relied on the roll "not keeping" the linking value, which only the code can show. In this volume, the roll never "gets" that value in the first place. Checking that the roll is honest thus moves from trusting code to a mathematical check that the voter's own device can perform.

3. Implementation and Testing

Everything lives in the case study's apps (casestudies/senate/apps), written in standard Go with no external libraries (Table 3).

File	Role
blind.go	RFC 9474: Prepare, EMSA-PSS-ENCODE (SHA-384, MGF1), Blind, BlindSign (CRT, with a check that the result opens back), Finalize, Verify
token.go	The token message, the key set (senate.token.keys), and token checking at the machine
tokenkeys.go	The roll: creating and keeping keys in the sealed register, signing blinded values, and test hooks (what the roll sees; a tagging roll)
roll.go, register.go	Accept blinded values instead of issuing tokens; keys in the sealed register
box.go	Relays the key set once, and publishes it on the public log when -public is on
device.go	Checks tokens against the group's published key; test hook (tokens the machine sees)
voters.go	The voter's device (test): create, blind, unblind, verify; collect every token first, then vote in shuffled order

Table 3 — Parts added or changed

A new test scenario in round 1 (V16) has the roll and the voting machines keep everything they see, as if both meant to link, and then compares the two. It also has the roll of district c1 sign one candidate's request with an unpublished key. Rounds 0–5 and 2b of Volume 1 were then all rerun on blind tokens.

4. Results

4.1 Conformance to the standard

Using the salt and blinding value of the RFC 9474 test vector [3][6], the blinded message, the blinded signature and the final signature match the RFC byte for byte. The final signature verifies with Go's crypto/rsa; signatures under another key and on a modified message fail. Signing a blinded value takes about 1.3 milliseconds on a 2-core workstation, negligible next to the time an official takes to check an ID card.

4.2 The roll and the voting machines cannot link (RQ1)

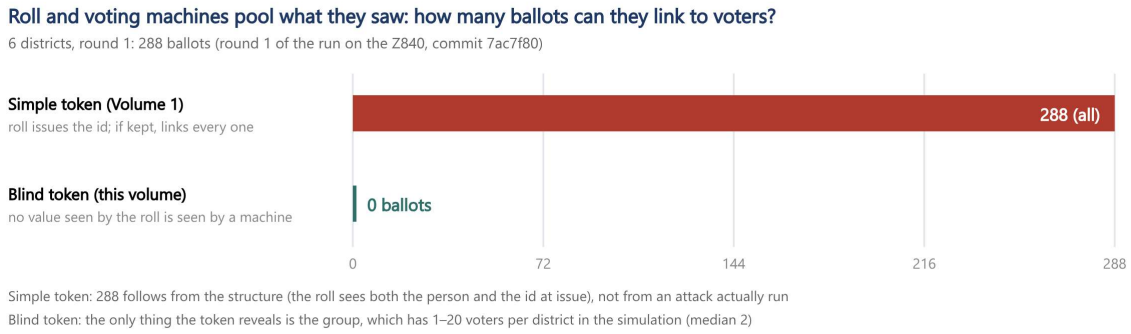


Figure 2 — Ballots the roll and the voting machines can link to voters when they pool their data

In round 1 the rolls of all 6 districts saw 289 requests (48 per district, plus 1 tagged request), each with the requester, the blinded value and the blinded signature. Every voting machine saw 288 tokens with their signatures. Comparing the two sets, no value appears in both. In the old design the roll sees the token's id at issue, so it can link all 288 by construction (Figure 2). This does not prove the cryptographic security, which comes from RFC 9474 and the original work [2][3], but it confirms that the app leaks no value connecting the two sides.

4.3 A roll that tags a voter is caught (RQ2)

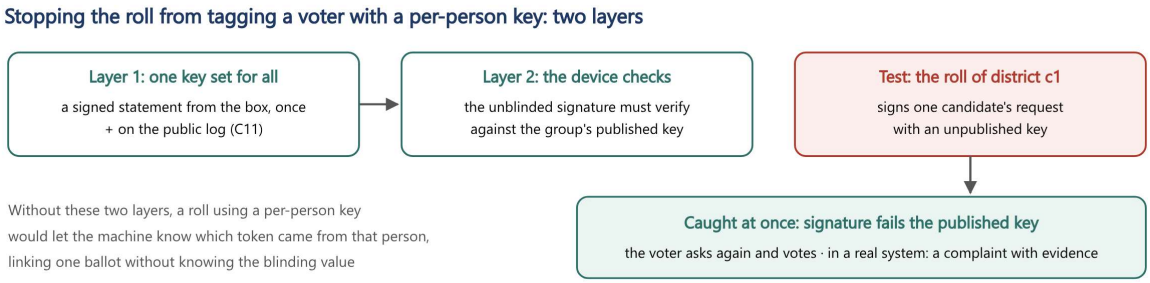


Figure 3 — Stopping the roll from tagging a voter with a per-person key

If the roll used a special key for one voter, the voting machine would know which token came from that person. The design stops this in two layers (Figure 3). First, the key set is a signed statement that everyone in the unit receives identically from the box, and it is on the public log. Second, the voter's device verifies the unblinded signature against the published key. In round 1 the roll of district c1 signed one candidate's request with an unpublished key; that voter's device caught it at once, because the signature did not verify against the published key. In the test the voter asked again and voted. In a real system this should become a complaint with evidence: the signature the roll sent and the signed key set.

4.4 The existing system works as before (RQ3)

Round	Result	Change from Volume 1
0 (run before every round, 6 times)	30/30 each	—
1 district level	28/28	V16 adds three checks (no linking, what the token reveals, a tagging roll)
2 lottery and three-level flow	25/25	Provincial and national rolls use their own keys
2b public log	24/24	106 public statements (20 key sets added), all verified by an observer without a certificate
3 national level, 3,000 people	9/9	All 3,000 voters obtained blinded tokens
4 inquiry	28/28	The 3,000-person register, with its keys, is sealed; after a restart it needs a grant first
5 asis versus heain	11/11	—

Table 4 — Rounds 0–5 and 2b on blind tokens (Z840, commit 7ac7f80, 30 Oct 2026)

Every round passed with no change to core, 155 checks in all. The results of the watch, the buyer and the inquiry are the same as in Volume 1, because blind tokens change only how the right to vote is

obtained, not the ballot or the count. On the workstation, round 1 voting at national level for 3,000 people took 49 seconds (previously 27), because it now includes requesting blinded tokens, still tiny next to a real selection day of about 21 hours [1].

4.5 What the token still reveals (RQ4)

The token reveals the voter's unit, round and occupational group, because keys are separate per group. In round 1 of the simulation, a group in a district had 1–20 voters (median 2). Real figures are similar: 45,753 eligible voters in 928 districts and 20 groups average about 2.5 per group per district [5]. The anonymity of the blind token is therefore bounded by the group, the same scope as ballots already sealed per group. In a group with a single voter, no method can hide whose ballot it is. Provincial and national levels have more voters per group (about 150 per group nationally), so they benefit more from this idea.

5. Discussion

5.1 What the idea closes and what it does not

Problem	Blind token	Note
Roll keeps requester–token pairs	closed	The roll never sees the token
Roll and voting machines collude	closed	Needs the blinding value from the voter's device
Roll tags voters with special keys	detected	Requires publishing the key set and having devices check
Reusing a token, or getting two	closed (as before)	Each id is used once; the roll issues once per round
Groups with very few voters	not closed	Bounded by group size, not by the signature scheme
Timing correlation	reduced	Tokens first, then voting in shuffled order; a real polling site needs further design
Vote buying by photographing the screen or ballot	not closed	A matter of result disclosure and the polling site (Volume 1)
Voter's device compromised or leaking the blinding value	not closed	Devices must be trustworthy, or use cards issued by an agency separate from the roll

Table 5 — The scope of the blind token

5.2 The voter's device

The idea requires voters to have a device that can hold the blinding value, such as a phone or a smart card. The device must generate good randomness, not leak the blinding value, and verify signatures against the published key. In the Senate selection the voters are the candidates themselves,

numbering in the tens of thousands, so issuing devices is more feasible than in a general election. An alternative is a token kiosk at the polling site, separate from the roll, which needs no personal device but requires trusting that the kiosk and the roll do not collude. The author chose the first option for this case study.

5.3 Relation to other cases

The same mechanism applies to elections of members of the House of Representatives, where each polling unit has hundreds of voters, so anonymity within a unit is much broader. It also applies to any task needing a single-use right without revealing identity, such as an anonymous survey. The limits concerning devices and the polling site remain the same. In every case core need not know this signature scheme: the existing lane separation, sealing and public log are enough to build the idea on.

6. Conclusion and Recommendations

Blind tokens close the gap Volume 1 left open. The roll checks eligibility and issues one token per person per round as before, but never sees the token, and even if the roll and the voting machines pool their data they cannot link voters to ballots. A roll that tries to tag a voter is caught. All of this is done in the app with no change to core, and the existing system of Volume 1 passes every round.

1. **If digital selection is to be trialled,** use blind tokens instead of simple tokens from the start, and always publish the key set on the public log.
2. **Define a complaint procedure** for when a voter's device finds a signature that does not match the published key, with verifiable evidence.
3. **Study voter devices:** phones, smart cards and kiosks separate from the roll, including device certification.
4. **State the group-size limit clearly.** At district level there are about 2.5 voters per group on average, so ballot secrecy at this level has a natural limit whatever method issues the token.

References

[1] Boonma, J. "The 2024 Senate Selection Case Study on heain-core, Volume 1: Applying the HEAIN Framework to the Existing Selection System." 30 October 2026 (companion paper).

[2] Chaum, D. "Blind Signatures for Untraceable Payments." Advances in Cryptology: Proceedings of Crypto 82, Springer, 1983, pp. 199–203.

[3] Denis, F., Jacobs, F., and Wood, C. A. "RSA Blind Signatures." RFC 9474, IRTF, October 2023.

[4] Moriarty, K., Kaliski, B., Jonsson, J., and Rusch, A. "PKCS #1: RSA Cryptography Specifications Version 2.2." RFC 8017, IETF, 2016.

[5] Thai PBS. Numbers of candidates reporting and passing the district-level Senate selection. 10 June 2024. thaipbs.or.th/news/content/340854 (in Thai).

[6] Cloudflare. CIRCL: blindsign/blindrsa, testdata/test_vectors_rfc9474.json.gz (a copy of the RFC 9474 test vectors). github.com/cloudflare/circl

[7] Boonma, J. "docs/progress-log.md" and casestudies/senate/ in the heain-core repository, 2026; and the project document claude/decision-2026-10-30-blind-tokens.

Author and copyright holder: Mr. Jakkrit Boonma (Independent Researcher) | ORCID: 0009-0000-7165-3787 |
Email: jaggid.boonma@yahoo.co.th

© 2026 Jakkrit Boonma. All rights reserved. Ownership and copyright in this document belong to Mr. Jakkrit Boonma from the date of publication, under applicable copyright law.