REFERENCE SOURCE CODE IMPLEMENTATION

KNOWN ISSUES

APRIL 6, 2004

SCOPE

In the process of preparing this package for release, the coding practices and architecture of the VHTi reference implementation were reviewed for programming and cryptographic best practices. This document lists known issues in the following categories:

1. Functionality not yet implemented;
2. Issues to be resolved by the VHTi-integrated voting application;
3. Incomplete or incorrect documentation; and
4. Future feature requests.

FUNCTIONALITY NOT YET IMPLEMENTED

Input validation

It is common for callers to pass **NULL** for optional output parameters. So, output parameters should be minimally checked (i.e., non-**NULL**) before assigning values to these parameters.

Vote verification transcript generation

When the Trustees generate verification statements after the election, they perform a secret-key operation on the public election data (using the VHTi function **VHTI_generate_pre_verification_results**). In order to make this procedure publicly auditable, it must provide a zero-knowledge proof of correctness. The structure of this proof is identical to the one created in **VHTI_partial_decrypt.**

TO BE RESOLVED BY THE VOTING APPLICATION

Missing signatures

In order to make all election participants *indisputably* accountable for their respective roles, digital signatures must be produced and verified where appropriate. Signatures have been incorporated in several important places, but there are some critical structures that should be signed which currently are not. A couple of examples are Shuffle outputs and poll site VoteVerificationDictionaries.

OpenSSL random routines

Many functions make use of functionality from OpenSSL. In a few of those functions, OpenSSL is used to generate random numbers. Most examples of this are in the **crypt** library, for doing such things as generating DSA keys. In order to get good randomness, application developers should take care to seed the OpenSSL PRNG with some entropy before calling any of these VHTi functions.

DOCUMENTATION

Formal election transcript specification

In order to support a truly open standard for election verification, there must be a published standards document, a so-called "Election Transcript", that specifies exactly what public data every election *must* produce, and exhaustively itemizing every check that must be performed on this data. Although the vast majority of such a document is *implied* by the code, the protocols, and the samples (e.g., see src/examples/election_sample/post-election-audit.sh), a formal Election Transcript specification is still planned.

Inconsistencies between glossary and DTD

During development of this reference implementation, we have striven to maintain accurate, concise, and consistent documentation. However, there are still some inconsistencies between the glossary and DTD. Generally, the DTD will be updated to match the glossary. Some specific notational issues to be aware of are highlighted in chapter 2 of the *API Reference*.

Sensitive information

Currently there is no clear delineation between sensitive and public information. A reasonable rule of thumb is that any element with the word "Secret" in its name is probably secret, but there are exceptions. For example, by its very nature, "RandomState", should be kept secret.

FEATURE REQUESTS

Full multi-thread support

Full multi-threaded support is not implemented. The only currently known VHTi function that suffers from lack of multi-threaded support is **VHTI_get_last_error**. Since there is only a single, global last error value, it is possible for two simultaneously failing function calls to collide, with one of the error messages being lost, and the other one being delivered to whichever thread asks for it first.

Missing data elements in SignedBlankBallot

There may be some data elements that are at least desirable to have part of the SignedBlankBallot. Some or all of these may be influenced by jurisdiction policy. A couple of examples might be the string length of VoteVerificationCode and AlphabetEncoding. Such omissions are not significant and will be added as appropriate in subsequent releases.

PKI library

There is an implicit assumption that public keys are trusted when used to check signatures. This functionality can be implemented by the voting application, but in subsequent releases, VHTi will support countersigned or X.509 certificates.

Threshold scheme for Trustees

Codebook management requires *n* of *n* Trustee participation. This will be generalized to a Pedersen threshold key-sharing scheme now used for results tabulation.

Unchecked signatures

Two functions, **VHTI_check_vcode_partial_decrypts_and_combine** and **VHTI_generate_pre_verification_results** currently take SignedVotedBallots as input. These ballots are individually signed, but the VHTi library does not currently check the signatures. Depending on details of the signing, either the **voter roll** would be required or a list of voting machine public keys. For now, these signatures should be checked by the voting application using **VHTI_check_xml**.

VoteVerificationCodes Collisions

As currently implemented, a very small number of BSNs may receive the same VoteVerificationCode for more than one BallotAnswer assigned to *the same* BallotQuestion. If these BSNs are used, it is possible that the voter's VoteReceipt will be ambiguous.

Though this may appear to be a problem, the event probability is small enough that extremely high confidence can still be achieved via the protocol. A simple remedy would be for voters who receive colliding BSNs to "spoil" the BSN and ask for another. Subsequent versions of the VHTi library will implement VoteVerificationCode generation so as to completely eliminate collisions within the same BallotQuestion.

Write-in votes

Current write-in support consists of assigning one AnswerMark to the distinguished "*write-in*" BallotAnswer. This allows voters to verify that their vote was cast as *a* write-in, and at tabulation time, results verification can check that the *number* of write-in votes in the tally is the same as the number of voters who received write-in confirmation from voter verification.

However, it is up to the voting application to keep track of the actual *values* of the write-in fields, and it would therefore be possible for the application to change the value of one or more of these fields to something different from what was indicated by the voter without detection.

Some states, such as California, may deem a ballot candidate as a "pre-registered write in" candidate for election policy reasons. VHTi does not treat these candiidates as "write-in" candidates, and end-to-end election verification applies to them exactly as it does to any other ballot candidate or issue.