Timeline
Week 1 requirements analysis
Week 2-3 protocol design and analysis
Week 2-4 architecture and design specification, including API
Week 4-7 implementation, validation, and verification
Week 7-8 compilation to library and Javascript

Cost
The cost of the technical piece of this proposal that includes system development and internal project management of the Galois development project is $33,000. Our effort analysis of this architecture indicates approximately 6 person-weeks of work is necessary to design and build a high-assurance, open source product that fulfills these requirements. The cost of such work for us with no margin or room for error is $20,000, thus we propose to price this work at $33,000 to accommodate for changes in requirements, errors in effort estimates, etc.

Scope
The scope of this subcomponent of the project is precisely the analysis, design, development, validation, and verification of the secure high-assurance subsystem of the Ballot.ly project. More specifically, we will focus on the cryptographic protocol, API development, and implementation of the anonymous state server fulfilling the security and correctness requirements of the Ballot.ly system. We will produce three code artifacts: (1) a server deployable in a cloud setting, (2) a native library for use in the development of native mobile applications, and (3) a JavaScript library for use in a browser-based client. We will also produce non-code artifacts, including a formal specification and verification of a cryptographic network protocol, a formal system specification, a formally specified language-neutral network API, and independently-auditable evidence of the system's correctness and security. All produced artifacts will be open source.

Technical Solution

The overall Ballot.ly implementation has three components: a view module, through which all user interaction is conducted (there may be multiple implementations of this component, including native applications for desktop and mobile devices and a web-based implementation); a ballot style server, which stores information about all the available ballot styles and provides them to the view module upon request; and an anonymous state server, which maintains the minimal amount of state information about users' ballot sessions. All communication among these components takes place over encrypted Internet connections.

A key aspect of the implementation is that no personal information is ever stored in the system; a user must enter a valid address and choose a ballot language in order to find her correct ballot style, but this information is used only for that purpose and is never stored on a server. Sample ballot sessions involve only local processing by the view module and perform no Internet communication whatsoever. When a user suspends or completes a sample ballot session, the anonymous state server generates a "session code" that allows her to resume where she left off or to retrieve her completed sample ballot later. Actual user choices are never stored on any server, anywhere in the system, with one exception: if a user has listed any write-in candidates on her ballot, her write-in candidates' names (but not their ballot positions, what ballot style they were written into, or any indication of who wrote them in) are recorded by the anonymous state server for later retrieval.

A valid session code allows the view module to reconstruct a user's session using only ballot style information obtained from the ballot style server and, when applicable, write-in name information obtained from the anonymous state server. We intend for session codes to be easily readable; perhaps sequences of words, such as "whole-disturbed-dragon" or "fickle-poor-pastry", rather than long strings of digits or computer-readable barcodes. Such human-readable encodings are already used in various security contexts, such as the PGP cryptosystem[1] and S/KEY authentication,[2] as order confirmation "numbers" on e-commerce sites such as meh.com, and in other settings.

This implementation strategy provides both usage flexibility and information security. A user can suspend her Ballot.ly session at any time and resume it later on any other Ballot.ly-compatible device, anywhere in the world; the only way private information about her ballot choices can be compromised is if somebody else obtains her session code.

The technical tasks associated with Galois's effort on this project are:

1. define the APIs and cryptographic protocols for the native and JavaScript libraries (to be used in native and web-based view modules)
2. define the communication protocols to be used among the components, including their security aspects
3. determine the encoding method to be used for session codes
4. design and develop the native and JavaScript libraries; we intend to generate these from a common code base
5. design and develop the anonymous state server

We expect that bandwidth requirements for the anonymous state server will typically be low, that bandwidth requirements for the ballot style server will typically be moderate, and that both servers will see large spikes of activity as Election Days approach. Thus, the natural choice for deployment is a public cloud service that can respond appropriately to elastic demand, such as Microsoft Azure or Amazon EC2. Our security architecture, and in particular the fact that personal information is never stored on any Ballot.ly server, allows us to host Ballot.ly using a public cloud service with no risk of personal data compromise.

[1] "Pretty Good Privacy", developed by Phil Zimmermann in 1991. Human-readable representations, originally designed to be shared by individuals during telephone calls, can be used for PGP key "fingerprints". An overview of PGP is available at http://en.wikipedia.org/wiki/Pretty_Good_Privacy.
[2] The S/KEY One-Time Password System, described in RFC 1760 (https://www.ietf.org/rfc/rfc1760.txt), uses six-word sequences taken from a dictionary of 2048 short (1–4 letter) words as an encoding for one-time passwords.