Shareable Crypto Wallet

Kyle Martin (ksmartin@bu.edu) & Cristian Morales (crism@bu.edu)
EC544 Project Technical Report | Wednesday, April 29th, 2020
Github: https://github.com/kylesmartin/ShareableWallet

PROJECT GOAL

The goal of this project was to create a system in which two users could transmit cryptocurrency funds between them using encoded images.

More specifically, a user sending funds would use this system to create a set of standard Electrum wallets which would then serve as cosigners for a multisig Electrum wallet. Once funds had been placed into the multisig wallet, a directory containing information about the multisig wallet and one of its cosigners would be created. This directory would then be AES-256-CBC encrypted, and the AES password would then be encrypted with the RSA public key of the Bitcoin funds' intended recipient. The encrypted directory and password would be converted to a text file, called a 'steg', which is embedded into a randomly-selected image. An encoded image could then be created for each multisig-cosigner pair, and the images could be sent to the intended recipient of the Bitcoin funds.

The user receiving the funds would have already used the system to generate the RSA key pair used to encrypt the stegs, and an Electrum wallet to receive the funds with. After receiving the encoded images, they can use the system to extract the RSA- and AES-encrypted steg from each image. The stegs can then be decrypted and the multisig and cosigner information therein be placed into appropriate files. Finally, the system can use the multisig and cosigner files to transfer (and approve the transaction as each cosigner) the Bitcoin funds from the multisig into the recipient's own Electrum wallet,

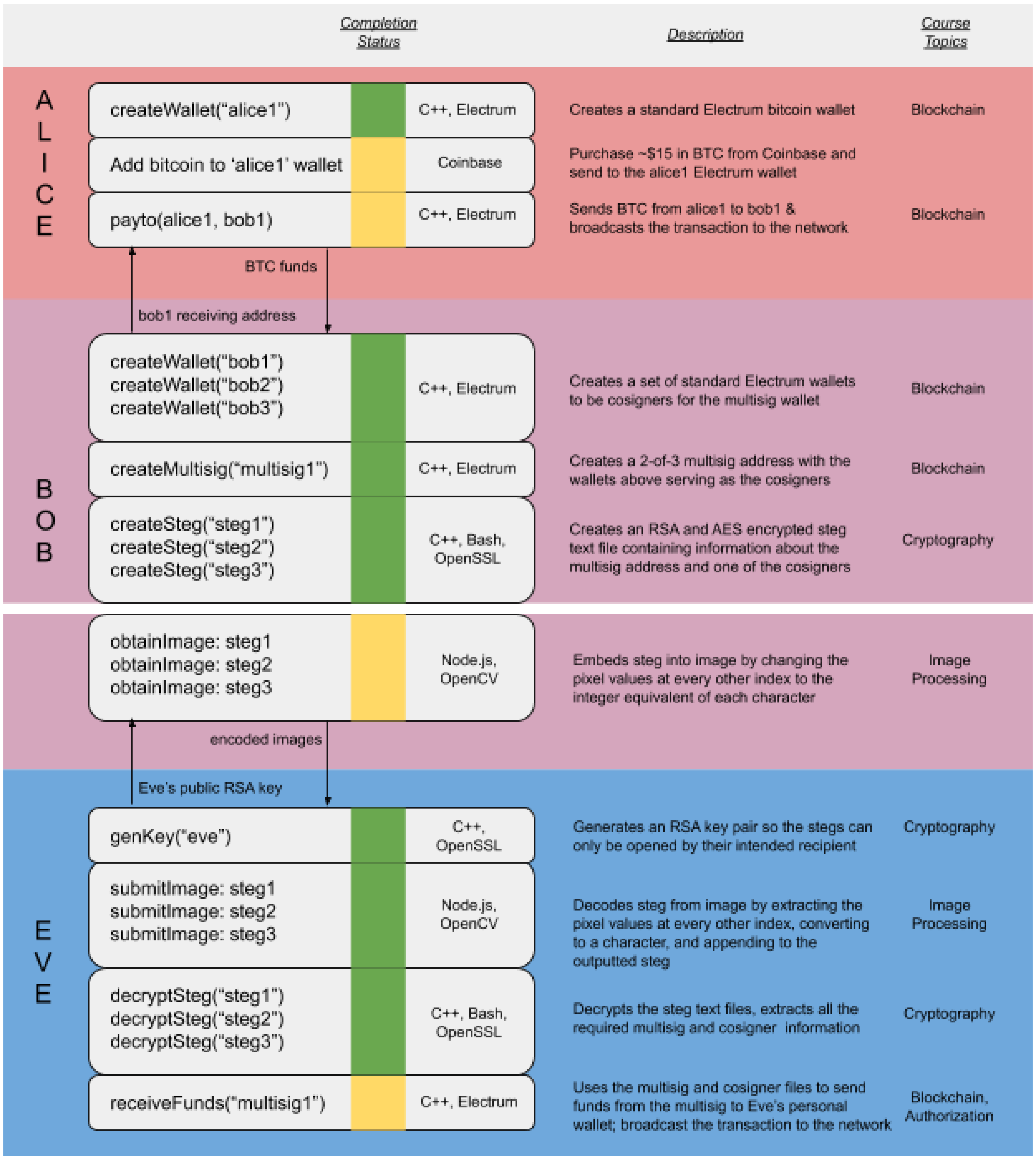
PROJECT RESULTS

The project was implemented moderately successfully, with most areas being fully successful and a few areas having technical failures. (These technical failures were never outright failures though, as the desired functionalities were there but were just not working consistently and in all cases.)

The only aspects of the project proposal that was not implemented in the final system is the use of the Lorem Picsum API to select the image being encoded with a steg. This API call was not implemented because the encoding program required that the image be stored locally on the server. As an alternative, we stored 100 images from Lorem Picsum onto the server and randomly selected one each time.

One piece of the final system we had not originally intended to implement was the CLI prompts for the user to interact and select system actions with. Originally, the system ran as a single executable, but we soon realized that this sometimes led to user confusion and that user input was required to set files names.

SYSTEM DIAGRAM



		Completion Status		Description	Course Topics
ALICE	createWallet("alice1")	🟩	C++, Electrum	Creates a standard Electrum bitcoin wallet	Blockchain
	Add bitcoin to 'alice1' wallet	🟨	Coinbase	Purchase ~$15 in BTC from Coinbase and send to the alice1 Electrum wallet	
	payto(alice1, bob1)	🟨	C++, Electrum	Sends BTC from alice1 to bob1 & broadcasts the transaction to the network	Blockchain

BOB	createWallet("bob1")
	createWallet("bob2")
	createWallet("bob3")
		🟩	C++, Electrum	Creates a set of standard Electrum wallets to be cosigners for the multisig wallet	Blockchain

	createMultisig("multisig1")	🟩	C++, Electrum	Creates a 2-of-3 multisig address with the wallets above serving as the cosigners	Blockchain

	createSteg("steg1")
	createSteg("steg2")
	createSteg("steg3")
		🟩	C++, Bash, OpenSSL	Creates an RSA and AES encrypted steg text file containing information about the multisig address and one of the cosigners	Cryptography

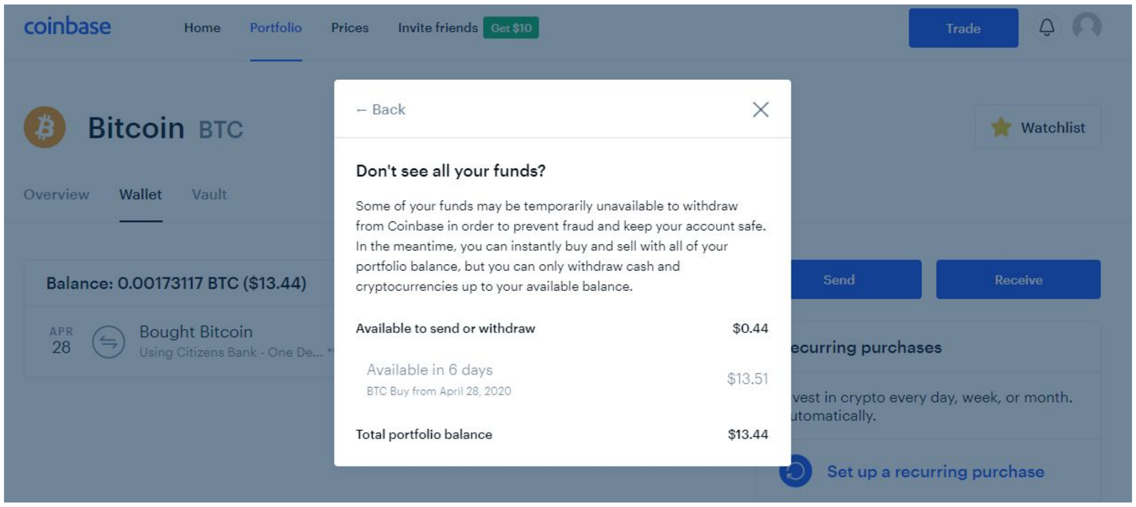
	obtainImage: steg1 obtainImage: steg2 obtainImage: steg3	🟨	Node.js, OpenCV	Embeds steg into image by changing the pixel values at every other index to the integer equivalent of each character	Image Processing
EVE	genKey("eve")	🟩	C++, OpenSSL	Generates an RSA key pair so the stegs can only be opened by their intended recipient	Cryptography
	submitImage: steg1 submitImage: steg2 submitImage: steg3	🟩	Node.js, OpenCV	Decodes steg from image by extracting the pixel values at every other index, converting to a character, and appending to the outputted steg	Image Processing

	decryptSteg("steg1")
	decryptSteg("steg2")
	decryptSteg("steg3")
		🟩	C++, Bash, OpenSSL	Decrypts the steg text files, extracts all the required multisig and cosigner information	Cryptography

	receiveFunds("multisig1")	🟨	C++, Electrum	Uses the multisig and cosigner files to send funds from the multisig to Eve's personal wallet; broadcast the transaction to the network	Blockchain, Authorization

DISCUSSION OF FAILURES

- **Adding bitcoin to a wallet:** We did actually purchase some Bitcoin funds so they could be introduced into and move through our system to demonstrate its success with a real-life cryptocurrency. However, we did not know in advance that funds can be purchased immediately for a personal Coinbase account, but will be held for a week before being able to be sent to a Bitcoin address. Because of this, we couldn't send our Bitcoin funds to any of the Electrum wallets being used for our project. This had ramifications discussed below in *payto()* and *receiveFunds()*.



Balance: 0.00173117 BTC ($13.44)

APR 28 Bought Bitcoin
Using Citizens Bank - One De...

← Back

Don't see all your funds?

Some of your funds may be temporarily unavailable to withdraw from Coinbase in order to prevent fraud and keep your account safe. In the meantime, you can instantly buy and sell with all of your portfolio balance, but you can only withdraw cash and cryptocurrencies up to your available balance.

Available to send or withdraw	$0.44
Available in 6 days BTC Buy from April 28, 2020	$13.51
Total portfolio balance	$13.44

ecurring purchases

vest in crypto every day, week, or month.
utomatically.

- **payto():** Because we had Bitcoin funds but were unable to add them into our system, all the Electrum wallets created by our programs had a balance of 0. And because Electrum/Bitcoin doesn't allow for transactions of 0 between empty wallets, our system was unable to create or approve transactions, thereby limiting our ability to demonstrate the functionality of this part of the system works as coded. We investigated altering the program to operate on the 'testnet' Bitcoin block chain, but Electrum's functionality with the testnet block chain was too limited to make that a viable solution.

- **obtainImage:** The encoding function was fully functional with text files that contained smaller strings. However, when using the text files that contained stegs, the images were not being generated. This was because the steg strings were too long to be encoded into the image using the greatest common denominator of the image height and width as the pixel offset. Because the number of pixels ran out before the number of characters in the steg, the stop condition is never met and the image is never generated. Hardcoding the offset to two pixels solved the problem on one of our local machines, but did not solve it consistently enough on the other.

- **receiveFunds():**The *receiveFunds()* function is intended to create a transaction between the multisig wallet and a recipient wallet, and then to automatically approve the transaction as the multisig cosigners. So although parts of this function work, the parts relying on *payto()* above do not, so we again could not demonstrate/verify its functionality.

DISCUSSION OF SUCCESSES

- **createWallet():** Standard Electrum wallets are successfully created and all the relevant information about a wallet is stored in a set of text files.
- **createMultisig():** A set of Electrum wallets can be used to generated a multisig wallet which they serve as cosigners of, and the user can specify the multisig threshold number
- **createSteg():** The information files on a multisig wallet and a cosigner are successfully compressed into a tarball which is then AES- and RSA-encrypted and converted to a Base64 text file for image encoding.
- **genKey():** The system can consistently automate the generation of an RSA public-private key pair.
- **submitImage:** Given that the images are properly encoded, this site will store uploaded images on the server, run the decoder on the image, and send a text file back to the client for download.
- **decryptSteg():** Stegs in Base64 text file formats are converted to an encrypted tarball which is RSA- and AES-decrypted and the multisig and cosigner files therein are decompressed.

Working on this project helped us gain a new set of skills and knowledge:

- **Cristian:** I have a pretty good theoretical understanding of how the Bitcoin blockchain works now and know how to practically manage/operate a Bitcoin wallet. (I also now own Bitcoin for the first time, which is pretty cool, the week delay in moving it notwithstanding.) I'm also now very comfortable using OpenSSL and encrypting/decrypting files using RSA and AES.
- **Kyle:** This project helped me develop a better understanding of good coding practice in node.js. This included using npm init to start projects, using a "public" folder for any static files, using a "views" folder for any ejs or html files, and being sure to save imports into the "node_modules" folder so that the node project can be run on any machine. I also learned about the practice of steganography and how to encode a message into an image without increasing the size of the file. Finally, I learned new ways to transfer information from server to client and client to server, such as file uploading with Multer.

REFERENCES

- Bitcoin Electrum - Unofficial guides for Electrum
- Bitzuma: A Beginner’s Guide to the Electrum Bitcoin Wallet
- Coinbase
- Official Electrum Documentation
- OpenSSL, Manpages for 1.1.1 - Commands