HACKERMONTHLY

Issue 45 February 2014

黃國富

Curator
Lim Cheng Soon

Contributors
Naval Ravikant
Brennan Moore
Alex MacCaw
Thomas Fuchs
Philip Zimmermann
Steven Lott
Brian Hayes
Andrew "bunnie" Huang
Sean Cassidy
Cecily Carver
John H. Lienhard
Andrew Rossignol
Oleg Andreev
Saar Drimer

Proofreaders
Emily Griffin
Sigmarie Soto

Illustrator
Jaime G. Wong

Ebook Conversion
Ashish Kumar Jha

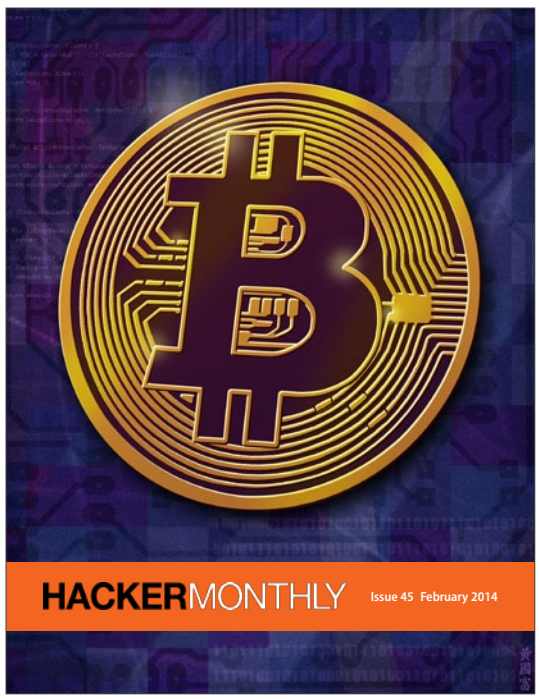
Printer
MagCloud

HACKER MONTHLY is the print magazine version of Hacker News — *news.ycombinator.com*, a social news website wildly popular among programmers and startup founders. The submission guidelines state that content can be "anything that gratifies one's intellectual curiosity." Every month, we select from the top voted articles on Hacker News and print them in magazine format. For more, visit *hackermonthly.com*

Advertising
ads@hackermonthly.com

Contact
contact@hackermonthly.com

Published by
Netizens Media
46, Taylor Road,
11600 Penang,
Malaysia.



Cover Illustration: Jaime G. Wong

Contents

FEATURES

STARTUPS

SPECIAL

PROGRAMMING

For links to Hacker News dicussions, visit *hackermonthly.com/issue-45*

Bitcoin — The Internet of Money

By NAVAL RAVIKANT

黃國富

Bitcoin will eventually be recognized as a platform for building new financial services.

Most people are only familiar with (b)itcoin, the electronic currency, but more important is (B)itcoin, with a capital B, the underlying protocol, which encapsulates and distributes the functions of contract law.

Bitcoin encapsulates four fundamental technologies:

- Digital Signatures — these can't be forged and allow one party to securely verify a transaction with another.
- Peer-to-Peer networks, like BitTorrent or TCP/IP — difficult to take down and no central trust required.
- Proof-of-Work prevents users from spending the same money twice, without needing a central authority to distinguish valid from invalid transactions. Bitcoin creates an incentive for miners, who run powerful computers in the network, to validate transactions and to secure them from future tampering. The miners are paid by "discovering" new coins, and anyone with computational resources can anonymously and democratically become a miner.
- Distributed Ledger — Bitcoin puts a history of each and every transaction into every wallet. This "block chain" means that anyone can validate that a given transaction was performed.

Thanks to these technical underpinnings, bitcoins are scarce (Central Banks can't inflate them away), durable (they don't degrade), portable (can be carried and transmitted electronically or as numbers in your head), divisible (into trillionths), verifiable (through everyone's block chain), easy to store (paper or electronic), fungible (each bitcoin is equal), difficult to counterfeit (cryptographically impossible), and can achieve widespread use — many of the technologists that brought us advances on the Internet are now working overtime to improve Bitcoin.

Proponents of the role of government argue that a currency with fixed supply will fail. They posit that inflation is required to keep people spending and that prices and wages are still as sticky as they were decades ago. They overlook that the world functioned on fixed money supplies until 40 years ago (the gold standard), and that bitcoin can gather many uses and value long before it has to become the main currency in which all prices are denominated. Another fear is that a central actor could take over the Bitcoin computing network — but the combined Bitcoin distributed supercomputer runs at the equivalent of 2,250 PetaFLOPS, 90x the rate of the fastest supercomputer (note — in Nov, it's now 48,000 PetaFLOPS!), and consumes an infinitesimal fraction of the resources used by a bloated banking system. Many label it as a speculative pyramid scheme — without realizing that all government-printed money is such. To the extent anyone holds cash over other assets, they are speculating that other assets will decline in relative value. Concerns abound over the security of the encryption scheme, the speed of transactions, the size of the block chain, the irreversibility of the transactions, and the potential for hacking and theft. All are fixable through third-party services and protocol upgrades. It's better to think about Bitcoin the protocol as Bitcoin 1.0, destined to evolve just as HTTP 1.0 evolved beyond of simple text and image-only web-browsers.

So why not just use Pounds or Dollars? One can use bitcoins as high-powered money with distinct advantages. Bitcoins, like cash, are irrevocable. Merchants don't have to worry about shipping a good, only to have a customer void the credit card transaction and charge-back the sale. Bitcoins are easy to send — instead of filling forms with your address, credit card number, and verification information, you just send money to a destination address. Each such address is uniquely generated for that single transaction, and therefore easily verifiable. Bitcoins can be stored as a compact number, traded by mere voice, printed on paper, or sent electronically. They can be stored as a passphrase that exists only in your head! There is no threat of money printing by a bankrupt government to dilute your savings. Transactions are pseudonymous — the wallets do not, by default, have names attached to them, although transaction chains are easy to trace. It has near-zero transaction costs — you can use it for micropayments, and it costs the same to send 0.1 bitcoins or 10,000 bitcoins. Finally, it is global — so a Nigerian citizen can use it to safely transact with a US company, no credit or trust required.

> **“Just as the web democratized publishing and development, Bitcoin can democratize building new financial services.”**

Even more importantly, Bitcoin the protocol will enable financial services transactions that are not possible today or require expensive and powerful third-parties.

Bitcoin has a scripting language which enables more than a “send money from X to Y” transaction. A Bitcoin transaction can require M of N parties to approve a transaction. Imagine wills that automatically unlock when most of the heirs agree that their parent has passed, no lawyer required. Or business accounts that require two of any three trusted signatures to approve an expenditure. Or wire escrows that go through when any arbiter agrees that the supplier sent the goods to the buyer. Or wallets that are socially secured by your friends and family. Or an allowance account accessible by the child and either of two parents. Or a crowdfunding of a Kickstarter project that pays out on milestones, based on the majority of the backers approving the next payment. The escrow in each case can be locked so that the arbiters can’t take the money themselves — only approve or deny the transaction.

The scripting language can also unlock transactions based on other parameters. Unlocking them over time can enable automatic mortgage, trust, and allowance payouts. Unlocking them on guessable numbers creates a lottery auditable by third parties. One can even design smart property — for example, a car’s electronic key so that when and only when a payment is made by the car buyer to the seller, the seller’s car key stops working and the buyer’s car key (or mobile phone) starts the car. Imagine your self-driving car negotiating traffic, paying fractional bitcoin to neighboring cars in exchange for priority.

Everyone has a copy of the Bitcoin block chain, so anyone can verify your transactions. You can write software that will crawl the block chain and generate automatic accounting histories for tax and verification purposes. You can engage in “Trusted Timestamping” — take a cryptographic signature of any document, timestamp it, and put it into the block chain. Anyone can verify that the document existed at a given time. If you sign the document with your private key and another party signs it with theirs, it becomes an undeniable, mutually signed contract. This entirely eliminates notaries, and websites like *proofofexistence.com* are showing the concept. The Namecoin project is building a distributed Domain Name System that allocates and resolves Domain Names without needing ICANN or Verisign, by using the block chain to establish proof-of-ownership. Similarly, look for entrepreneurs to apply this authoritative proof-of-ownership to build P2P Stock and Bond Exchanges — at least one Bitcoin site, “Satoshi Dice,” has sold shares and issues dividends without using a stock exchange. The ownership and dividends are easily verifiable by anyone who wants to look inside the block chain. Predictious.com is combining the transaction scripting and the verifiability to create a prediction market in which you cannot be cheated and third-party arbiters can allocate the winnings.

Bitcoin’s “send-only” and irreversible nature makes it much less vulnerable to theft. Today, anyone with your Credit Card or E-Checque (ACH) information can pull money from your account. This creates chargebacks, expensive dispute resolution and merchants double-checking your identity. Bitcoin is send only. Anyone who has received bitcoins from you can’t request or pull more money from your account.

Most importantly, Bitcoin offers an open API to create secure, scriptable e-cash transactions. Just as the web democratized publishing and development, Bitcoin can democratize building new financial services. Contracts can be entered into, verified, and enforced completely electronically, using any third-party that you care to trust, or by the code itself. For free, within minutes, without possibility of forgery or revocation. Any competent programmer has an API to cash, payments, escrow, wills, notaries, lotteries, dividends, micropayments, subscriptions, crowdfunding, and more. While the traditional banks and credit card companies lock down access to their payments infrastructure to a handful of trusted parties, Bitcoin is open to all.

Silicon Valley knows a platform when it sees it, and is aflame with Bitcoin. Teams of brilliant young programmers, entranced by the opportunity, are working on Exchanges (Payward, Buttercoin, Varum), Futures Markets (ICBIT), Hardware Wallets (BitCoinCard, Trezor, etc), Payment Processors (*bitpay.com*), Banks, Escrow companies, Vaults, Mobile Wallets, Remittance Networks (*bitinstant.com*), Local Trading networks (*localbitcoins.com*), and more.

Looming over them is how governments view Bitcoin and the entrenched financial powers it threatens. The last few decades have seen a move towards a cashless society, where every transaction is tracked, reported, and controlled. Bitcoin takes powers from the central actors and returns it to merchants and consumers, savers, and borrowers. Bitcoin brings back some pseudonymity in the transactions, and can be irrevocably traded like cash. And finally, it points a way towards a single currency — it is a bug, not a feature, that we have multiple global currencies with exchangers and transaction fees in between.

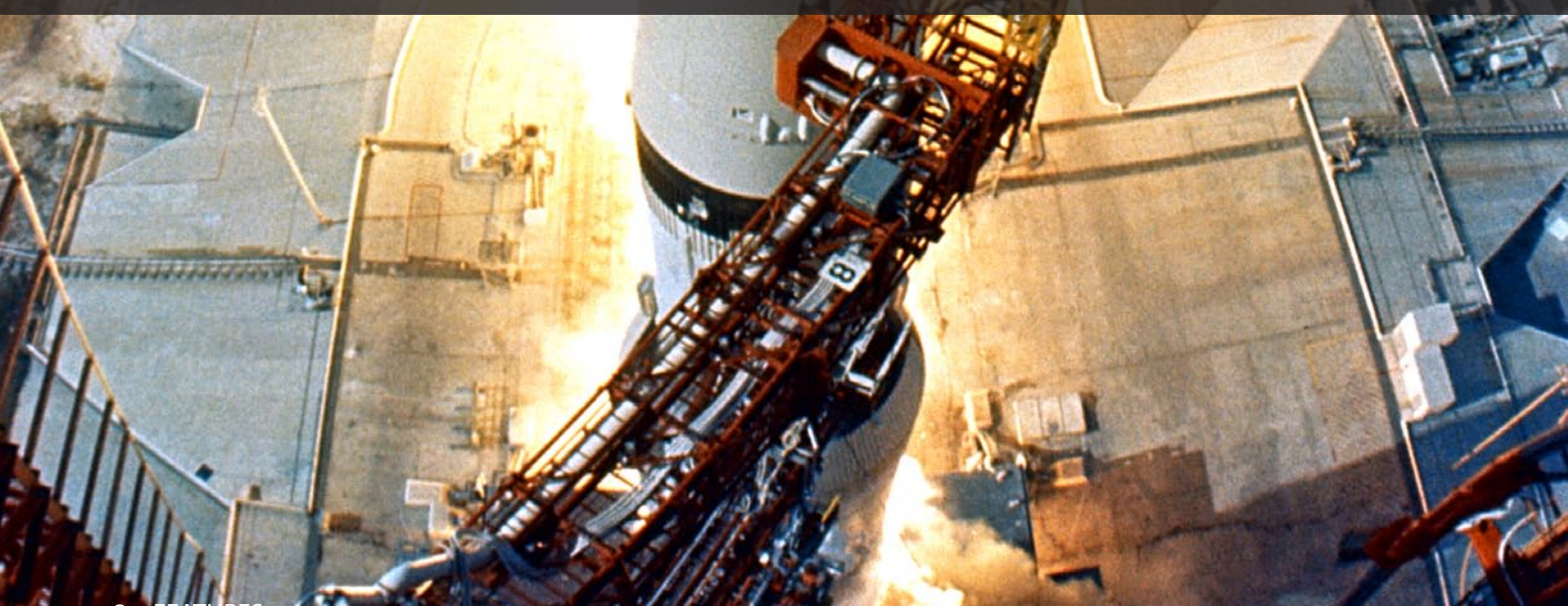
Governments have been cracking down on the bitcoin exchanges, making it harder to obtain and slowing its development. Strict and expensive Money Transmitter regulations, designed to slow terrorist and child porn financing, threaten the next great technological revolution — never mind that terrorists can use cash just fine, the means of terror are cheap, and that they account for an infinitesimal fraction of global commerce. The development and innovation in Bitcoin has already begun the move to friendlier jurisdictions, where its innovation can continue un-impeded. Regulators in the US and UK would be wise to proceed with a light touch, lest they push the development of Bitcoin and its entrepreneurs to places like Canada, Finland, and the Sino-sphere. The United States has benefited enormously from being home to the majority of global companies driving the Internet revolution. The country that is the home to the Internet of Money could one day end up as the guardian of the new Reserve Currency and the Global Money Supply. ■

Naval Ravikant is the CEO and a co-founder of AngelList. He previously co-founded Epinions (which went public as part of Shopping.com) and Vast.com. Naval is an active Angel investor, and have invested in dozens of companies, including Twitter, Uber, Yammer, Stack Overflow and Wanelo. Follow him on Twitter at *@naval*

Reprinted with permission of the original author.
First appeared in *hn.my/bitcoin* (startupboy.com)

Debugging a Live Saturn V

By BRENNAN MOORE



We all have stories, as engineers, of fixing some crazy thing at the last minute right before the demo goes up. We have all encountered situations where we needed to fix something that was our fault and we needed to fix it now.

This story is something that I think about in those times to remember to stay calm. No last minute fix could ever be this dramatic or important.

My grandfather passed away about a week ago. At the service, I was asked to say a few words and read from his memoirs. This was my choice.

****

Red Team 4 To The Pod

The first unmanned launch of a Saturn V on November 9th. 1967. From the personal memoirs and the pen of William E. Moore January 28th. 1994.

There was five of us Rocket Scientists lounging around the ready room listening to the Apollo 4 Countdown on loud speakers and headsets. We were members of the Red Team Group and we were the Electrical Systems experts on all hardware interfaces between the firing room and the Saturn V vehicle three miles away. Our ears were now being drawn into a developing situation happening on the net. No response was received from an electrical circuit that controlled the separation of the S-II Stage from the S-1C Stage in flight.

"That was one of my electrical circuits!"

It just so happened that circuit is controlled by a series of relays located almost directly beneath that cold beast that was spewing out all kinds of funny colored, very cold gases — the Saturn V rocket. We took a look at our blue prints and found the relay that must be the problem and called for a recycle in the countdown to a point where we could cycle the switch on the electrical networks console to see if the relay would pick up — that was a "no go". Now things got serious. The NASA Test Conductor was talking "scrub the launch" but our S-II Stage Test Conductor was talking "go to the pad".

Well, the Red Phone rings.

"Bill, how sure are you that this relay is the problem? Are we going to send people to the pad to rewire the rocket and not be able to launch because we guessed wrong?" said "AC" Filbert C. Martin

"It's worth a shot, the signal is not reaching the vehicle and that relay module is the only active component between the Firing Room Console and the Vehicle. You snap out the old Relay Module and snap in the new one and we will be able to tell if that was the problem a few seconds later."

"Well, we are a little concerned about sending a team to the pad with a fully loaded vehicle. We thought your team would do a lot of blueprint trouble shooting — I'm not sure we planned to actually send anybody out to a fueled vehicle."

"Just don't let them launch this mother till we are at least half way back from the pad — OK!"

About thirty minutes later the five of us (Bob Kelso NRR Sr. Tech, Bill Moore; NAR Engineer/ Team Leader, the NASA Safety Engineer, the NRR Quality Control and the NASA Pad Leader) got the official word to head for the Launch Pad with our new Relay Pod. It was 11:30pm. It was a dark, slow, three mile trip. As we got closer to the Saturn V it was shrouded in a white cloud of venting gases which relieved the pressures building up inside the vehicle fuel tanks.

Our goal was to enter this two level hermetically sealed, all welded steel coffin called the Mobil Launcher Base topped by a fully loaded 363 ft. high Saturn V, weighing 6.2 million pounds, and the permanently attached 380 ft. high Umbilical Tower, weighing 500,000 pounds. We finally stopped and left our van to walk up and into the second level of the Mobile Launcher Base. About this time, it came to my mind that during one of our training sessions we were told that one of the fully fueled prototype S-II rocket stages had been exploded out in the desert. The results showed that all buildings better be at least three miles from the launch pads - which they are. We were now within 25 feet of this 363ft tall bomb that sounded like its giant fuse had been lit, and we were soon going to get much closer.

The Saturn V was more noisy and ghostly than I had ever expected and it had grown much taller and certainly more threatening since last week. The venting fuel made loud hissing sounds when relief valves popped or opened up suddenly. It was very easy to let your imagination infect your brain. This is a very dangerous place and

everything seems to be moving in the heavy foggy mist. There was no way to talk to each other, heck, we could barely see each other and... we hadn't thought of this problem so we held onto each other's yellow protective clothing like kindergartners crossing the street. We all wore safety helmets but they just did not make you feel like you were really safe.

We climbed up the last step prior to opening the sealed submarine type entry door that led into the second level. We slowly opened the heavy steel hatch-type pressurized door and it was like stepping into the jaws of a huge steaming dragon. The nitrogen fog, used to suppress fire, and the dim red glow from the emergency lights of level A made it look like a hollywood swamp scene. We started making our way through the 21 compartments to find our Relay Rack as the noise took on a more penetrating tone that seemed to bounce from wall to wall.

The smell became a mixture of kerosene with a mild touch of burnt paint and rubber. I was glad that the astronauts did not take this path to go aboard the Saturn V because my goosebumps were changing to a weird color of purple. With the realization that this was a much worse place to be trapped in, the team moved more rapidly to the relay rack. We replaced the old relay module and then had to cycle the switch on the firing room console. We then checked that the relay kicked in and that the signal was picked up on the vehicle. We resealed the cabinet, signed off on all the paperwork and got the out of there without any more sightseeing.

The drive back to the ready room very was fast and uneventful. The five of us were like stone figures, thinking about where we had been and what we had just accomplished. What could have happened and didn't. All of this without ever realizing that this experience was as close to being in the shoes of a Saturn V astronaut as any of us would ever be again.

****

In later letters, my grandfather mentions how fortunate he really was, having growing up a farm boy in West Virginia to have not just once-in-a-lifetime experiences, but really once-in-many-lifetimes experiences. The service was about celebrating his life, and this seems like one of those incredibly unique events that really does celebrate his life, both in terms of how he handled a mind bogglingly stressful situation and how he tells it so comfortably detailed and with slyly humorous ease that was so characteristic of how he spoke.

A really incredible man who really contributed a lot to the world around him and meant a lot to those close to him, he will be sorely missed. ■

Brennan Moore leads the web team at *Artsy.net*, a site for discovering and collecting art. He is looking for Bojangles biscuits in Brooklyn.

Reprinted with permission of the original author.
First appeared in *hn.my/saturnv* (zamiang.com)

Now you can hack on DuckDuckGo

DuckDuckHack

Create instant answer plugins for DuckDuckGo

duckduckhack.com

An Engineer's Guide to Stock Options

By ALEX MACCAW

There's a lot of fear, uncertainty and doubt when it comes to stock options, and I'd like to try and clear some of that up today. As an engineer, you may be more interested in getting on with your job than compensation. However, if you're working at a fast growing startup, with a little luck and the right planning you can walk away from a liquidity event with a significant amount of money.

On the other hand I have friends who have literally lost out on millions of dollars because the process of exercising stock options was so complicated, opaque and expensive. Believe me, you'll be kicking yourself if this happens to you, so why not arm yourself with some knowledge and make informed decisions?

This guide is an attempt to correct some of the imbalance in information between companies and employees, and explain in plain English the whole stock option process.

Shares 101

I like thinking about shares as a virtual currency. Shareholders are speculating on that currency, and the company is trying to increase its value. Companies can inflate or deflate this currency depending on their performance and perceived potential or by issuing new shares.

When companies are formed, they typically issue around 10 million shares. These are split between members of the founding team and are diluted in subsequent investment rounds. A portion of these shares are put aside into an option pool, a group of shares dedicated for employees. Any shares you receive will probably come out of this pool.

Stock Options

When you join a company, you probably won't receive any shares though, but rather the option to buy shares. This is a contract which states you have the "option" to buy shares at a specific price.

You can think of a stock option as a Future. The company is basically saying: "Here's our current valuation. We hope it'll go up. In a year or so, once you've worked at the company for a while, we'll give you the option to buy shares in the company at the price when you joined, even if our valuation has subsequently increased."

Vesting schedules

Option agreements typically have a four-year vesting schedule, with a one year cliff. In plain English this means that you will receive all your stock options over a period of four years, but if you leave in less than a year (or are fired) then you won't receive any options at all.

The "cliff" is included to incentivize employees to stay at least a year, and to protect the company's shareholders if the founders decide that you're not a good fit.

Typically you see your shares broken up into 1/48ths. You get 12/48 at your 1 year mark, and each month after that you'll vest another 1/48.

Exercising

Once you've cliffed, you have the right to buy shares in the company. There are few ways in which you can benefit from this right:

- **Acquisition:** Hope that the company is acquired and the shares are sold at a large multiple of the exercise price in your option agreement. Investors pay a premium and their shares are preferred for a reason — if the company is sold for less than the value placed on it at the last round of investment, your shares will probably be worth next to nothing.
- **Secondary market:** Stock option agreements usually give the company a right of first refusal. This means that you cannot sell the shares to a third party without giving the company the opportunity to buy them first. However, once a company reaches a certain stage, the board may allow you to sell your shares through an exchange like Second Market or some other mechanism. At this stage you can cash-out by selling your vested shares to outside investors.
- **Cashless exercise:** In the event of an IPO, you can work with a broker to exercise all of your vested options and immediately selling a portion of them into the public market. This means you can afford both the shares, and the tax without having to invest money yourself.
- **Exercise before leaving:** You can write your company a check and pay any taxes due — in return, you'll get a stock certificate and become a shareholder in the company. You can carry on working at the company (and exercise more shares as they vest) or leave whenever you want.
- **Exercise after leaving:** You leave the company, and send a check for all your vested shares before 90 days is up. This, combined with a cashless exercise, are probably the two most common scenarios.

Each route has different tax implications that can depend on the timing of the sale and the amounts involved. As a general rule, if the company you're working for is growing like crazy (and you think it might go public someday) it makes a lot of sense to exercise your right to become a shareholder as soon as possible.

Depending on your personal financial situation, the number of options granted to you, their exercise price, and their change in value, exercising the right to buy all of your vested shares may be prohibitively expensive.

Even if you have the cash, you may not want to spend your life savings on a stock certificate and a tax bill. The earlier you joined the company, the cheaper these shares will have been. If the value of those shares have increased considerably there will be significant tax liabilities. Furthermore you'll probably only make money on the investment if there's a liquidity event. This is why early employees at fast-growing startups essentially have a pair of golden handcuffs on and cannot leave — they're paper millionaires but they're not able to exercise their right to buy the shares and therefore have to stick around until the company is sold or goes public.

If you decide that you want to leave (and you think the company has a great future ahead) you typically have about 90 days to decide whether you want to exercise your vested shares and come up with the cash to buy the shares and the associated taxes. If you can't afford to exercise, or decide not take the risk, then the option expires.

Questions you should ask going in

When you join a company, there are some important questions you should ask:

- How many shares will I have the option to exercise?
- How many shares are there in outstanding? (Or what is the total number of shares?)
- What is the exercise price per share? (Or what price can I buy them for?)
- What is the preferred share price? (Or what have investors paid for their shares)?
- What does my vesting schedule look like?

These questions will let you figure what it would cost to buy the shares and the current valuation of the company. Crucially, you'll be able to calculate the percentage of the company your shares would represent if they were all vested today. As a company grows and issues more shares this percentage will decrease as your shares are diluted. Nevertheless, it's still good to have a rough idea of the percentage of the company you own when you start.

Don't be deceived if you're offered a large number of shares without any mention of the number of shares currently outstanding. Many companies are reluctant to share this kind of information and claim it's confidential.

If the company seems reluctant to answer these questions, keep pressing and don't take "no" for an answer. If you're going to factor in your options into any compensation considerations, you deserve to know what percentage of the company you're getting, and its value.

I'd be wary of compromising on salary for shares, unless you're one of the first few employees or founders. It's often a red flag if the founders are willing to give up a large percentage of their company when they could otherwise afford to pay you. Sometimes you can negotiate a tiered offer, and decide what ratio of salary to equity is right for you.

Likewise, I'd take into consideration the likelihood of an IPO when estimating how valuable your options are. Companies such as small consultancies or lifestyle businesses may offer you shares, but a return is unlikely. Having a small slice of ownership may feel good, but may ultimately be worthless. If the company has been around for a few years without a clear upward trajectory, an IPO is probably unlikely.

Other questions

There are some other questions you should ask, but may have a harder time getting a straight answer:

- How many shares is the company authorized to issue?
- Have any shares been issued with a liquidation preference greater than 1x?

The answers to these questions could affect any returns. For example, if the company dilutes the stock pool, then the value of your shares will decrease. Additionally, if some investors have a preferred liquidation preference, then they have the right to cash out first if there's a liquidity event.

Example scenario #1

Let's say the company gives you the option to buy 100,000 shares at an exercise price (or strike price) of $0.50 per share. If the company has 10,000,000 shares outstanding, then you have the option to buy 1% of the company when fully vested. It also means the current valuation of the company is $5 million USD.

Let's say you leave the company after the first year, meaning you have only vested 25,000 shares (100,000 / 4), which will cost you $12,500 USD to purchase. This is a highly simplified example which doesn't include any tax liabilities, but it gives you the general idea.

409A valuations & tax

A 409A valuation is a fair market valuation of a company as determined by an accountant and is reported to the IRS. This valuation is often lower than the valuation at the last investment round because investors are more optimistic about the company's future, and are speculating on its potential. As a company approaches an IPO, the delta between these two valuations will shrink and eventually disappear.

By comparing the company's 409A valuation when you were granted the options and the 409A valuation when you purchase the stock, you can get a good indication of your tax liabilities. If you've only been at the company for a year (or the company hasn't grown materially), the 409A valuation may not have changed, and if you decide to buy shares you'll have no tax liability.

However, if the difference is significant, the IRS will treat this gain as an "AMT preference", and tax you on the spread. The tax bill can often be greater than the check you have to write to your company. You have to pay real money for gains that only exist on paper. What's more, if the company fails then you don't get the tax refunded — only credits towards your next tax return. This can substantially increase the risk on the investment.

The last thing worth mentioning here is that if you're buying vested shares before you leave the company, then I strongly suggest you look into filing a "83(b) election", which could significantly decrease the amount of tax you have to pay. A full explanation of 83(b) elections is a guide in itself, but essentially they let you pay all your tax liabilities for both vested and un-vested stock early, at the current 409A valuation (even if the valuation subsequently increases).

Things you should know going out

If you're thinking about leaving and you haven't bought any shares, you should decide whether or not you'd like to become a shareholder. If you think the company's going to be wildly successful, then it might be worth the risk. Assuming you decide to go ahead and purchase the stock, you have three months to give the company a check.

Ideally, you should know the following:

- How many shares have been issued
- The current 409A valuation
- Preferred share price of the last round

It's much easier to find out the answers to these questions when you're still at the company, so I suggest you get this information before you leave if at all possible.

Example scenario #2

You've left the company after a year and decided that you want to become a shareholder. Your option to buy the shares will expire 90 days after you've stopped working at the company, so you have to get the money together and give the company a check before that date. You know that the 409A valuation has increased from $0.50 a share to $5. Since you have the option to buy 25,000 shares (you vested a quarter of your 100,000 shares), this is going to cost you $12,500 to purchase.

However, since the 409A valuation of the company has increased to $5, the IRS will see the current valuation of your stock as $125,000, and you'll have to report a gain of $112,500 ($125,000 - $12,500). As income, this will be taxed at around 40% (~20% federal, ~20% state). Obviously this tax level will vary vastly between individuals, but let's just take 40% for argument's sake.

So your total cost to exercise is $12,500 to the company, and $45,000 to the government for a total of 25,000 shares.

Financing options

If you can't afford to exercise your right to buy your vested shares (or don't want to take the risk) then there's no need to despair — there are still alternatives. There are a few funds and a number of angel investors who will front you all the cash to purchase the shares and cover all of your tax liabilities. You hold the shares in your name and if there's a liquidity event you distribute a percentage of the profits to them. They'll typically ask for somewhere between 20-50% of the upside depending on the company, the taxes, and the size of the investment. It's an interest-free loan without a personal guarantee. If the company fails, you don't owe anyone anything; if it succeeds, you'll be rewarded for the value you created whilst working there.

If you're interested in learning more about financing your stock options then send me an email and I'll make some introductions. I've set up an informal mailing list, and have a group of angel investors subscribed who do these kinds of deals all the time.

Conclusion

The reason why I wrote this guide is that engineers are often the unsung heroes at startups, and they too deserve to benefit from the upside in any value they create. It's also why I'm excited about stock option financing, which serves to level the playing field a bit and make exercising affordable, whilst removing the risk for the engineer.

****

Thanks to Richard Burton, Colin Regan, Adam Fraser, Josh Buckley, Kip Kaehler, Tim O'Shea, and Andrew McCalister for helping with drafts of this article.
If you have questions or feedback, then feel free to email me [alex@alexmaccaw.com].

As with all information on the internet, take this with a pinch of salt and get advice from a professional CPA before making any decisions. None of this article is to be construed as legal or financial advice. ■

Alex MacCaw is an O'Reilly author, software engineer, traveller and founder of *sourcing.io*

Reprinted with permission of the original author. First appeared in *hn.my/stock* (alexmaccaw.com)

5 Things I've Learned in 5 Years of Running a SaaS

By THOMAS FUCHS

Freckle Time Tracking [letsfreckle.com] is turning five on December 1. In 5 years of being a co-founder of Freckle I've learned a lot of things, but here are 5 important takeaways. Maybe it'll helps you on your path to product nirvana:

1 You're not a "tech company" — you're a "make customers awesome" company

People don't pay you because you have amazing programming skills and can write nginx configurations blindfolded. People pay you money because the product you sell saves them time, money, effort and nerves. It's your job to make your customer more awesome. Every decision you make for your product and business should revolve around that.

2 Never promise dates for a feature launch

Just don't promise launch dates for a feature. Ever. Trust me on this. People will ask you all the time when "feature X" is ready. A good way to answer that question is (if you plan on doing it), "We're considering this feature for a future version. I can't give you a date on when it will be ready." Just be honest with your customers — you don't know yourself if and when a feature will really be ready.

3 Spend money on things that help you stay productive

This includes obvious stuff like a laptop that doesn't suck (upgrade often), a good working chair and desk, and less obvious things, like software that allows you to concentrate on developing your application's features rather than configuring servers.

4 Do not work too much

Overworking yourself is the first step to failure in business. You can't do your best if you're permanently stressed out. Don't check email in the evenings. If you're only 1 or 2 people, don't provide 24/7 support. It's ok. Customers understand. It helps to not have a mission-critical product (if Time Tracking goes down it's annoying, but people can make a note on paper).

You didn't start a company to die of exhaustion. Your health, family and social life is more important than 5 minute support response times and a 100% uptime guarantee.

By the way, one way to keep on top of this is to keep track on how you spend your time.

5 Don't believe the hype

People are good at getting excited. And people are good at believing the hype™ about new technologies, frameworks, programming languages and ways to deploy. People will tell you what to do and what to plan for. That you need to scale to millions of users, and that you're doomed if you don't plan for that. That generating HTML on the server is so 1994. That node.js will cure cancer.

The fact is that you need to be pragmatic — your goal is to run a business. Use technology that is proven (to you), and that you know how to work with. My "litmus test" for technology is if the people that provide it are in a similar situation as you are: having to rely on it to run their own business. You need to optimize for shipping. That includes writing less code, having broad test coverage, and concentrating on getting things out in order of long-term profitability for your business. ■

Thomas Fuchs was born and raised in Vienna, Austria and is now living in Philadelphia. He is the author of Zepto.js and script.aculo.us and a Ruby on Rails core alumnus. With his wife Amy Hoy he's building cheerful software, like Freckle Time Tracking [letsfreckle.com], Every Time Zone [everytimezone.com] and occasionally writes books like *Retinafy.me*

Reprinted with permission of the original author.
First appeared in *hn.my/5saas* (mir.aculo.us)

Why I Wrote PGP

By PHILIP ZIMMERMANN

> **"Whatever you do will be insignificant, but it is very important that you do it." — Mahatma Gandhi**

It's personal. It's private. And it's no one's business but yours. You may be planning a political campaign, discussing your taxes, or having a secret romance. Or you may be communicating with a political dissident in a repressive country. Whatever it is, you don't want your private electronic mail (email) or confidential documents read by anyone else. There's nothing wrong with asserting your privacy. Privacy is as apple-pie as the Constitution.

The right to privacy is spread implicitly throughout the Bill of Rights. But when the United States Constitution was framed, the Founding Fathers saw no need to explicitly spell out the right to a private conversation. That would have been silly. Two hundred years ago, all conversations were private. If someone else was within earshot, you could just go out behind the barn and have your conversation there. No one could listen in without your knowledge. The right to a private conversation was a natural right, not just in a philosophical sense, but in a law-of-physics sense, given the technology of the time.

But with the coming of the information age, starting with the invention of the telephone, all that has changed. Now most of our conversations are conducted electronically. This allows our most intimate conversations to be exposed without our knowledge. Cellular phone calls may be monitored by anyone with a radio. Electronic mail, sent across the Internet, is no more secure than cellular phone calls. Email is rapidly replacing postal mail, becoming the norm for everyone, not the novelty it was in the past.

Until recently, if the government wanted to violate the privacy of ordinary citizens, they had to expend a certain amount of expense and labor to intercept and steam open and read paper mail. Or they had to listen to and possibly transcribe spoken telephone conversation, at least before automatic voice recognition technology became available. This kind of labor-intensive monitoring was not practical on a large scale. It was only done in important cases when it seemed worthwhile. This is like catching one fish at a time, with a hook and line. Today, email can be routinely and automatically scanned for interesting keywords, on a vast scale, without detection. This is like driftnet fishing. And exponential growth in computer power is making the same thing possible with voice traffic.

Perhaps you think your email is legitimate enough that encryption is unwarranted. If you really are a law-abiding citizen with nothing to hide, then why don't you always send your paper mail on postcards? Why not submit to drug testing on demand? Why require a warrant for police searches of your house? Are you trying to hide something? If you hide your mail inside envelopes, does that mean you must be a subversive or a drug dealer, or maybe a paranoid nut? Do law-abiding citizens have any need to encrypt their email?

What if everyone believed that law-abiding citizens should use postcards for their mail? If a nonconformist tried to assert his privacy by using an envelope for his mail, it would draw suspicion. Perhaps the authorities would open his mail to see what he's hiding. Fortunately, we don't live in that kind of world, because everyone protects most of their mail with envelopes. So no one draws suspicion by asserting their privacy with an envelope. There's safety in numbers. Analogously, it would be nice if everyone routinely used encryption for all their email, innocent or not, so that no one drew suspicion by asserting their email privacy with encryption. Think of it as a form of solidarity.

Senate Bill 266, a 1991 omnibus anticrime bill, had an unsettling measure buried in it. If this nonbinding resolution had become real law, it would have forced manufacturers of secure communications equipment to insert special "trap doors" in their products, so that the government could read anyone's encrypted messages. It reads, "It is the sense of Congress that providers of electronic communications services and manufacturers of electronic communications service equipment shall ensure that communications systems permit the government to obtain the plain text contents of voice, data, and other communications when appropriately authorized by law." It was this bill that led me to publish PGP electronically for free that year, shortly before the measure was defeated after vigorous protest by civil libertarians and industry groups.

The 1994 Communications Assistance for Law Enforcement Act (CALEA) mandated that phone companies install remote wiretapping ports into their central office digital switches, creating a new technology infrastructure for "point-and-click" wiretapping, so that federal agents no longer have to go out and attach alligator clips to phone lines. Now they will be able to sit in their headquarters in Washington and listen in on your phone calls. Of course, the law still requires a court order for a wiretap. But while technology infrastructures can persist for generations, laws and policies can change overnight. Once a communications infrastructure optimized for surveillance becomes entrenched, a shift in political conditions may lead to abuse of this new-found power. Political conditions may shift with the election of a new government, or perhaps more abruptly from the bombing of a federal building.

A year after the CALEA passed, the FBI disclosed plans to require the phone companies to build into their infrastructure the capacity to simultaneously wiretap 1% of all phone calls in all major U.S. cities. This would represent more than a thousand-fold increase over previous levels in the number of phones that could be wiretapped. In previous years, there were only about a thousand court-ordered wiretaps in the United States per year, at the federal, state, and local levels combined. It's hard to see how the government could even employ enough judges to sign enough wiretap orders to wiretap 1 percent of all our phone calls, much less hire enough federal agents to sit and listen to all that traffic in real time. The only plausible way of processing that amount of traffic is a massive Orwellian application of automated voice recognition technology to sift through it all, searching for interesting keywords or searching for a particular speaker's voice. If the government doesn't find the target in the first 1 percent sample, the wiretaps can be shifted over to a different 1 percent until the target is found, or until everyone's phone line has been checked for subversive traffic. The FBI said they need this capacity to plan for the future. This plan sparked such outrage that it was defeated in Congress. But the mere fact that the FBI even asked for these broad powers is revealing of their agenda.

Advances in technology will not permit the maintenance of the status quo, as far as privacy is concerned. The status quo is unstable. If we do nothing, new technologies will give the government new automatic surveillance capabilities that Stalin could never have dreamed of. The only way to hold the line on privacy in the information age is strong cryptography.

You don't have to distrust the government to want to use cryptography. Your business can be wiretapped by business rivals, organized crime, or foreign governments. Several foreign governments, for example, admit to using their signal's intelligence against companies from other countries to give their own corporations a competitive edge. Ironically, the United States government's restrictions on cryptography in the 1990s have weakened U.S. corporate defenses against foreign intelligence and organized crime.

The government knows what a pivotal role cryptography is destined to play in the power relationship with its people. In April 1993, the Clinton administration unveiled a bold new encryption policy initiative, which had been under development at the National Security Agency (NSA) since the start of the Bush administration. The centerpiece of this initiative was a government-built encryption device, called the Clipper chip, containing a new classified NSA encryption algorithm. The government tried to encourage private industry to design it into all their secure communication products, such as secure phones, secure faxes, and so on. AT&T put Clipper into its secure voice products. The catch: At the time of manufacture, each Clipper chip is loaded with its own unique key, and the government gets to keep a copy, placed in escrow. Not to worry, though — the government promises that they will use these keys to read your traffic only "when duly authorized by law." Of course, to make Clipper completely effective, the next logical step would be to outlaw other forms of cryptography.

The government initially claimed that using Clipper would be voluntary, that no one would be forced to use it instead of other types of cryptography. But the public reaction against the Clipper chip was strong, stronger than the government anticipated. The computer industry monolithically proclaimed its opposition to using Clipper. FBI director Louis Freeh responded to a question in a press conference in 1994 by saying that if Clipper failed to gain public support, and FBI wiretaps were shut out by non-government-controlled cryptography, his office would have no choice but to seek legislative relief. Later, in the aftermath of the Oklahoma City tragedy, Mr. Freeh testified before the Senate Judiciary Committee that public availability of strong cryptography must be curtailed by the government (although no one had suggested that cryptography was used by the bombers).

The government has a track record that does not inspire confidence that they will never abuse our civil liberties. The FBI's COINTELPRO program targeted groups that opposed government policies. They spied on the antiwar movement and the civil rights movement. They wiretapped the phone of Martin Luther King, Jr. Nixon had his enemies list. Then there was the Watergate mess. More recently, Congress has either attempted to or succeeded in passing laws curtailing our civil liberties on the Internet. Some elements of the Clinton White House collected confidential FBI files on Republican civil servants, conceivably for political exploitation. And some overzealous prosecutors have shown a willingness to go to the ends of the Earth in pursuit of exposing sexual indiscretions of political enemies. At no time in the past century has public distrust of the government been so broadly distributed across the political spectrum, as it is today.

Throughout the 1990s, I figured that if we want to resist this unsettling trend in the government to outlaw cryptography, one measure we can apply is to use cryptography as much as we can now while it's still legal. When use of strong cryptography becomes popular, it's harder for the government to criminalize it. Therefore, using PGP is good for preserving democracy. If privacy is outlawed, only outlaws will have privacy.

It appears that the deployment of PGP must have worked, along with years of steady public outcry and industry pressure to relax the export controls. In the closing months of 1999, the Clinton administration announced a radical shift in export policy for crypto technology. They essentially threw out the whole export control regime. Now, we are finally able to export strong cryptography, with no upper limits on strength. It has been a long struggle, but we have finally won, at least on the export control front in the US. Now we must continue our efforts to deploy strong crypto, to blunt the effects increasing surveillance efforts on the Internet by various governments. And we still need to entrench our right to use it domestically over the objections of the FBI.

PGP empowers people to take their privacy into their own hands. There has been a growing social need for it. That's why I wrote it. ■

Philip R. Zimmermann is the creator of Pretty Good Privacy, an email encryption software package.

Reprinted with permission of the original author.
First appeared in *hn.my/pgp* (philzimmermann.com)

How to Design a Class

By STEVEN LOTT

1. Write down the words. You started to do this. Some people don't and wonder why they have problems.

2. Expand your set of words into simple statements about what these objects will be doing. That is to say, write down the various calculations you'll be doing on these things. Your short list of 30 dogs, 24 measurements, 4 contacts, and several "parameters" per contact is interesting, but only part of the story. Your "locations of each paw" and "compare all the paws of the same dog to determine which contact belongs to which paw" are the next step in object design.

3. Underline the nouns. Seriously. Some folks debate the value of this, but I find that for first-time OO developers it helps. Underline the nouns.

4. Review the nouns. Generic nouns like "parameter" and "measurement" need to be replaced with specific, concrete nouns that apply to your problem in your problem domain. Specifics help clarify the problem. Generics simply elide details.

5. For each noun ("contact", "paw", "dog", etc.) write down the attributes of that noun and the actions in which that object engages. Don't short-cut this. Every attribute. "Data Set contains 30 Dogs" for example is important.

6. For each attribute, identify if this is a relationship to a defined noun, or some other kind of "primitive" or "atomic" data like a string or a float or something irreducible.

7. For each action or operation, you have to identify which noun has the responsibility, and which nouns merely participate. It's a question of "mutability." Some objects get updated, others don't. Mutable objects must own total responsibility for their mutations.

8. At this point, you can start to transform nouns into class definitions. Some collective nouns are lists, dictionaries, tuples, sets or namedtuples, and you don't need to do very much work. Other classes are more complex, either because of complex derived data or because of some update/mutation which is performed.

Don't forget to test each class in isolation using unit test.

Also, there's no law that says classes must be mutable. In your case, for example, you have almost no mutable data. What you have is derived data, created by transformation functions from the source dataset. ■

Steven Lott is a consultant, teacher, author and software developer with over 35 years of experience building software of every kind, from specialized control systems for military hardware to large data warehouses.

Reprinted with permission of the original author.
First appeared in *hn.my/designclass* (stackoverflow.com)

The Keys to the Keydom

By BRIAN HAYES

Your security and privacy on the Internet (to the extent that such quaint notions still have meaning) depend on the difficulty of factoring certain large numbers. For example, take this 1,024-bit integer:

```
X = 12378451765455704420457203
00156476432601975715662027908
82488143432336664289530131607
57127360381500856200680250007 8
94557646372684708763884626821
07823064928561396351027680220 8
43687210122771845086073708050 2
11546298213957026549887480838 7
54401998419152234009077341926 5
57130978958668227065179495079 9
3107455319103401
```

The number printed above in squinty type is the product of two 512-bit prime factors. If you set out to find those factors, the project might well keep you busy for many megayears. But I can make the job much easier by giving you a second number of the same size:

```
Y = 13975280625857017971965733
49412654630082054150470733499
42370461270597321020717639292
87999215162641361024775042926 7
91623042401095505475050283351
70703959862897242371124108160 0
05581486237854115688455171463 0
34213840635250918248983182261 7
52341938159505970416275181409 0
63848892180548678870584294449 3
4835873139133193
```

Factoring both X and Y would appear to be twice as much work, but in fact you can do it lickety-split. On my laptop it took roughly 200 microseconds. From millions of years to millionths of a second — that's quite a speedup!

There's a trick, of course. Both X and Y are products of two large primes, but it so happens that one of the primes is a shared factor of both numbers. For finding that shared factor, we can rely on a very old, very famous, very simple and very efficient algorithm: Euclid's algorithm for the greatest common divisor. In Python it looks like this:

```
def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)
```

(The "%" in line 5 is Python's modulo or remainder operator.) When this function is applied to X and Y, the recursion is invoked 297 times before returning the common factor:

```
F = 10704679319376067064256301
45948715022696962191248959648
26285098009220803181996357261
17009340189103336170841315900 3
54200725312700639146605265442
630619090531
```

You don't have to take my word for it that F divides both X and Y. Do the division: In that way you will also learn the co-factors of X and Y.

If X and Y were components of public keys in the RSA cryptosystem, their shared factor would create a huge hole in the security fence. And the problem is particularly insidious in that each of the two keys, when examined in isolation, looks perfectly sound; the weakness only becomes apparent when you have both members of the pair.

This potential vulnerability of factoring-based encryption methods has been known for decades, but it seemed there was no reason to worry because coincidentally shared factors are so utterly unlikely. A couple of weeks ago I heard an eye-opening talk by Nadia Heninger, a member of a group that has searched for such unlikely coincidences in the wild. They found 64,000 of them. Reason to worry.

Heninger and her colleagues polled every public IPv4 address in the known universe, requesting a connection on the ports commonly used for two secure communication protocols, TLS and SSH. For every address that responded to queries on those ports, they collected the server's public encryption key and then closed the connection. Here I am going to discuss only the TLS servers with RSA keys; there were vulnerabilities in other cryptosystems as well, but the issues are slightly different.

Before telling the rest of this story, I have to pause here. For those of you in the born-digital generation, pinging every address on the Internet may sound like a routine walk around the block on

a sunny afternoon, but I confess that I never would have dared to try anything so audacious. It's like knocking on every door in America, or calling every possible telephone number — a task that's not feasible for individuals of ordinary means, and that also seems unforgivably rude. But standards of feasibility and rudeness are different in the world of machine-to-machine communication. Computers don't care if you make four billion hang-up calls (although some system administrators might frown on the practice). And, after all, the encryption keys being collected are by definition public.

Back to Heninger's story. They ran their scan of IP addresses from Amazon's Elastic Compute Cloud service, where the data-collection phase of the project took a few days. Out of $2^{32} \approx 4$ billion addresses (less a few special-purpose or reserved areas) they found about 29 million servers accepting connections on the standard port for TLS, but only 12.8 million of those servers supplied public keys. Some 60 percent of the keys retrieved were not unique. Presumably, most of the duplicates are accounted for by organizations that have multiple servers all operating with the same cryptographic credentials, but there were also instances of apparently unaffiliated individuals sharing a key. This is rather like discovering that your house key also opens your neighbor's front door (and vice versa).

After eliminating the duplicates, some 5.8 million distinct RSA keys needed to be tested for common factors. Even though Euclid's GCD algorithm is highly efficient, running it on all possible pairings of keys would be a strain. There's an ingenious shortcut, based on the observation that if Y is relatively prime to each of $X_1, X_2, \ldots, X_n$, then it also has no factor in common with the product $X_1 \times X_2 \times \cdots \times X_n$. Thus it's possible to detect the presence of shared factors with just n GCD operations, instead of n^2. A drawback of this approach is that the product of millions of RSA keys is a huge number, and intermediate results have to be swapped out to disk. Nevertheless, the processing was completed in an hour and a half on the Amazon cloud at a cost of $5.

The output was a list of 64,081 compromised keys for TLS hosts, about 0.5 percent of all such keys collected. For obvious reasons, Heninger et al. are not publishing that list; they tried to contact the owners of vulnerable machines, and they are also offering a web lookup service where you can check to see if your key is on the list.

The good news is that none of the weak keys are guarding access to major web servers hosting bank accounts or medical records or stock markets or military installations. Most of them are found in embedded networked devices, such as routers and firewalls. That's also the bad news. A programmer with malicious intent who can gain control of a well-placed router can make a lot of mischief.

Could the prevalence of common factors in RSA keys be explained as a product of pure bad luck? To answer this question we need to solve a birthday problem. The original version of this problem asks how many people you need to bring together before there's a good chance that two or more of them will have the same birthday (assuming birthdays are distributed randomly over the 365 days of the year). An order-of-magnitude approximation is $\sqrt{365}$, or about 19. (The actual number is 23.) For the RSA variant of the problem, we ask how many 512-bit primes you need to generate — assuming you select them uniformly at random from the set of all such primes — before you have a good chance of seeing at least one prime twice. In this case we replace 365 with the number of 512-bit primes, which is in the neighborhood of 10^{150}. Thus there's scarcely any chance of a collision until the number of randomly generated primes approaches 10^{75}. We're only at 10^7 so far. As Heninger said in her talk, we have enough 512-bit primes to assign a public encryption key to every atom in the universe, with little worry over possible duplicates.

According to this line of reasoning, it would be a colossal fluke to see even one duplicated RSA prime, and finding 64,000 of them is clear evidence that those primes are not being chosen uniformly at random. The blame apparently lies with pseudorandom number generators. It's not that the algorithms are defective. In many cases, cryptographic keys are being generated immediately after a machine is booted, when it just can't scrape together enough entropy to make a passable pseudorandom number. ■

Brian Hayes writes about math and computing for American Scientist magazine and for bit-player.org. And he is the author of Infrastructure: A Field Guide to the Industrial Landscape.

Reprinted with permission of the original author.
First appeared in *hn.my/keydom* (bit-player.org)

On Hacking MicroSD Cards

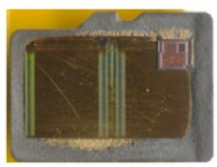
By ANDREW "bunnie" HUANG

Today at the Chaos Computer Congress (30C3), xobs [xoblo.gs] and I disclosed a finding that some SD cards contain vulnerabilities that allow arbitrary code execution — on the memory card itself. On the dark side, code execution on the memory card enables a class of MITM (man-in-the-middle) attacks, where the card seems to be behaving one way, but in fact it does something else. On the light side, it also enables the possibility for hardware enthusiasts to gain access to a very cheap and ubiquitous source of microcontrollers.

In order to explain the hack, it's necessary to understand the structure of an SD card. The information here applies to the whole family of "managed flash" devices, including microSD, SD, and MMC, as well as the eMMC and iNAND devices typically soldered onto the mainboards of smartphones and used to store the OS and other private user data. We also note that similar classes of vulnerabilities exist in related devices, such as USB flash drives and SSDs.

Flash memory is really cheap. So cheap, in fact, that it's too good to be true. In reality, all flash memory is riddled with defects — without exception. The illusion of a contiguous, reliable storage media is crafted through sophisticated error correction and bad block management functions. This is the result of a constant arms race between the engineers and mother nature; with every fabrication process shrink, memory becomes cheaper but more unreliable. Likewise, with every generation, the engineers come up with more sophisticated and complicated algorithms to compensate for mother nature's propensity for entropy and randomness at the atomic scale.

These algorithms are too complicated and too device-specific to be run at the application or OS level, and so it turns out that every flash memory disk ships with a reasonably powerful microcontroller to run a custom set of disk abstraction algorithms. Even the diminutive microSD card contains not one, but at least two chips — a controller, and at least one flash chip (high density cards will stack multiple flash die). You can see some die shots of the inside of microSD cards at a microSD teardown [hn.my/microtear] I did a couple years ago.



In our experience, the quality of the flash chip(s) integrated into memory cards varies widely. It can be anything from high-grade factory-new silicon to material with over 80% bad sectors. Those concerned about e-waste may (or may not) be pleased to know that it's also common for vendors to use recycled flash chips salvaged from discarded parts. Larger vendors will tend to offer more consistent quality, but even the largest players staunchly reserve the right to mix and match flash chips with different

controllers, yet sell the assembly as the same part number — a nightmare if you're dealing with implementation-specific bugs.

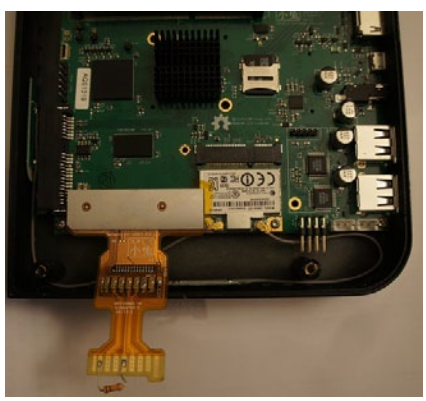
The embedded microcontroller is typically a heavily modified 8051 or ARM CPU. In modern implementations, the microcontroller will approach 100 MHz performance levels, and also have several hardware accelerators on-die. Amazingly, the cost of adding these controllers to the device is probably on the order of $0.15 – $0.30, particularly for companies that can fab both the flash memory and the controllers within the same business unit. It's probably cheaper to add these microcontrollers than to thoroughly test and characterize each flash memory chip, which explains why managed flash devices can be cheaper per bit than raw flash chips, despite the inclusion of a microcontroller.

The downside of all this complexity is that there can be bugs in the hardware abstraction layer, especially since every flash implementation has unique algorithmic requirements, leading to an explosion in the number of hardware abstraction layers that a microcontroller has to potentially handle. The inevitable firmware bugs are now a reality of the flash memory business, and as a result it's not feasible, particularly for third party controllers, to indelibly burn a static body of code into on-chip ROM.

The crux is that a firmware loading and update mechanism is virtually mandatory, especially for third-party controllers. End users are rarely exposed to this process, since it all happens in the factory, but this doesn't make the mechanism any less real. In my explorations of the electronics markets in China, I've seen shop keepers burning firmware on cards that "expand" the capacity of the card — in other words, they load a firmware that reports the capacity of a card is much larger than the actual available storage. The fact that this is possible at the point of sale means that most likely, the update mechanism is not secured.

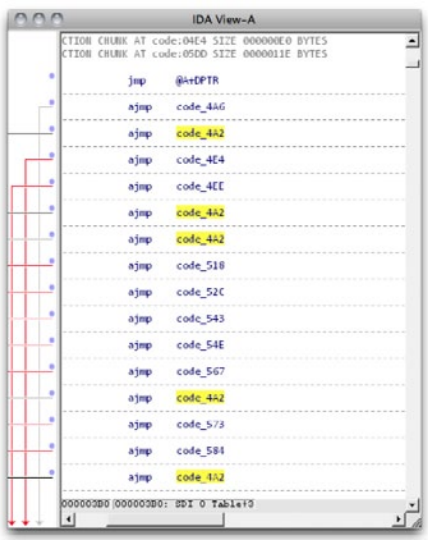
In our talk at 30C3, we report our findings exploring a particular microcontroller brand, namely, Appotech and its AX211 and AX215 offerings. We discover a simple "knock" sequence transmitted over manufacturer-reserved commands (namely, CMD63 followed by "A", "P", "P", "O") that drop the controller into a firmware loading mode. At this point, the card will accept the next 512 bytes and run it as code.

From this beachhead, we were able to reverse engineer (via a combination of code analysis and fuzzing) most of the 8051' s function specific registers, enabling us to develop novel applications for the controller, without any access to the manufacturer's proprietary documentation. Most of this work was done using our open source hardware platform, Novena, and a set of custom flex circuit adapter cards (which, tangentially, lead toward the development of flexible circuit stickers aka chibitronics).



Significantly, the SD command processing is done via a set of interrupt-driven call backs processed by the microcontroller. These callbacks are an ideal location to implement an MITM attack.

CMD INDEX	type	argument	resp	abbreviation	command description
CMD0	bc	[31:0] stuff bits	-	GO_IDLE_STATE	Resets all cards to idle state
CMD1	reserved				
CMD2	bcr	[31:0] stuff bits	R2	ALL_SEND_CID	Asks any card to send the CID numbers on the CMD line (any card that is connected to the host will respond)
CMD3	bcr	[31:0] stuff bits	R6	SEND_RELATIVE_ ADDR	Ask the card to publish a new relative address (RCA)
CMD4	bc	[31:16] DSR [15:0] stuff bits	-	SET_DSR	Programs the DSR of all cards
CMD5	reserved for I/O cards (refer to the "SDIO Card Specification")				
CMD7	ac	[31:16] RCA [15:0] stuff bits	R1b (only from the selected card)	SELECT/DESELECT _CARD	Command toggles a card between the stand-by and transfer states or between the programming and disconnect states. In both cases, the card is selected by its own relative address and gets deselected by any other address; address 0 deselects all. In the case that the RCA equals 0, then the host may do one of the following: - Use other RCA number to perform card deselection. - Re-send CMD3 to change its RCA number to other than 0 and then use CMD7 with RCA=0 for card deselection.
CMD8	bcr	[31:12]reserved bits [11:8]supply voltage(VHS) [7:0]check pattern	R7	SEND_IF_COND	Sends SD Memory Card interface condition, which includes host supply voltage information and asks the card whether card supports voltage. Reserved bits shall be set to '0'.
CMD9	ac	[31:16] RCA [15:0] stuff bits	R2	SEND_CSD	Addressed card sends its card-specific data (CSD) on the CMD line.
CMD10	ac	[31:16] RCA [15:0] stuff bits	R2	SEND_CID	Addressed card sends its card identification (CID) on CMD the line.
CMD11	reserved				
CMD12	ac	[31:0] stuff bits	R1b	STOP_ TRANSMISSION	Forces the card to stop transmission
CMD13	ac	[31:16] RCA [15:0] stuff bits	R1	SEND_STATUS	Addressed card sends its status register.
CMD14	reserved				



It's as of yet unclear how many other manufacturers leave their firmware updating sequences unsecured. Appotech is a relatively minor player in the SD controller world; there's a handful of companies that you've probably never heard of that produce SD controllers, including Alcor Micro, Skymedi, Phison, SMI, and of course, Sandisk and Samsung. Each of them would have different mechanisms

and methods for loading and updating their firmware. However, it's been previously noted that at least one Samsung eMMC implementation using an ARM instruction set had a bug which required a firmware updater to be pushed to Android devices, indicating yet another potentially promising venue for further discovery.

From the security perspective, our findings indicate that even though memory cards look inert, they run a body of code that can be modified to perform a class of MITM attacks that could be difficult to detect; there is no standard protocol or method to inspect and attest to the contents of the code running on the memory card's microcontroller. Those in high-risk, high-sensitivity situations should assume that a "secure-erase" of a card is insufficient to guarantee the complete erasure of sensitive data. Therefore, it's recommended to dispose of memory cards through total physical destruction (e.g., grind it up with a mortar and pestle).

From the DIY and hacker perspective, our findings indicate a potentially interesting source of cheap and powerful microcontrollers for use in simple projects. An Arduino, with its 8-bit 16 MHz microcontroller, will set you back around $20. A microSD card with several gigabytes of memory and a microcontroller with several times the performance could be purchased for a fraction of the price. While SD cards are admittedly I/O-limited, some clever hacking of the microcontroller in an SD card could make for a very economical and compact data logging solution for I2C or SPI-based sensors.

Slides from our talk at 30C3 can be downloaded here [hn.my/sdcard], or you can watch the talk on Youtube [hn.my/30c3]. ■

Andrew "bunnie" Huang loves hardware. He was involved in some of the earliest stages of hardware reverse engineering on the Xbox, and his experiences are summarized in his book, "Hacking the Xbox: An Introduction to Reverse Engineering". bunnie also serves as a Research Affiliate for the MIT Media Lab, technical advisor for several hardware startups and MAKE magazine, and shares his experiences manufacturing hardware in China through his blog.

Reprinted with permission of the original author.
First appeared in *hn.my/microsd* (bunniestudios.com)

Don't Pipe to your Shell

By SEAN CASSIDY

Piping wget or curl to bash or sh is stupid. Like this:

```
wget -O - http://example.com/install.sh | sudo sh
```

It's everywhere. Sometimes they tell you to ignore certificates as well (looking at you, Salt). That's dumb.

The main reason I think it's dumb (other than running arbitrary commands on your machine that could change based on user agent to trick you) is its failure mode.

What happens if the connection closes midstream? Let's find out.

```
(echo -n "echo \"Hello\""; cat) | nc -l -p 5555
```

This will send a command to whoever connects, but it won't send the newline. Then, it'll hang. Let's connect the client:

```
nc localhost 5555 | sh
```

At first, nothing happens. Great. What will happen if we kill -9 the listening netcat? Will sh execute the partial command in its buffer?

Yes.

```
nc localhost 5555 | sh
Hello
```

But what about wget, or curl?

```
wget -O - http://localhost:5555 | sh
--2013-10-31 16:22:38--  http://localhost:5555/
Resolving localhost (localhost)... 127.0.0.1
Connecting to localhost (local-
host)|127.0.0.1|:5555... connected.
HTTP request sent, awaiting response... 200 No
headers, assuming HTTP/0.9
Length: unspecified
Saving to: `STDOUT'

[       <=              ] 12    --.-K/s   in 8.6s

2013-10-31 16:22:47 (1.40 B/s) - written to
stdout [12]

Hello
```

What if that partial command wasn't a harmless echo but instead one of these:

```
TMP=/tmp
TMP_DIR=`mktemp`
rm -rf $TMP_DIR
```

Harmless, right? And what if the connection closes immediately after "rm -rf $TMP" is sent? It'll delete everything in the temp directory, which is certainly harmful.

This might be unlikely, but the results of this happening, even once, could be catastrophic.

Friends don't let friends pipe to sh. ■

Sean is a software engineer who is passionate about doing things right. He is currently working on Squadron [gosquadron.com]: an awesome configuration and release management tool for SaaS applications.

Reprinted with permission of the original author.
First appeared in *hn.my/pipeshell* (existentialize.com)

Things I Wish Someone Had Told Me When I Was Learning How to Code

And What I've Learned From Teaching Others

By CECILY CARVER

Before you learn to code, think about what you want to code

Knowing how to code is mostly about building things, and the path is a lot clearer when you have a sense of the end goal. If your goal is "learn to code," without a clear idea of the kinds of programs you will write and how they will make your life better, you will probably find it a frustrating exercise.

I'm a little ashamed to admit that part of my motivation for studying computer science was that I wanted to prove I was smart, and I wanted to be able to get Smart Person jobs. I also liked thinking about math and theory, and the program was a good fit. It wasn't enough to sustain me for long, though, until I found ways to connect technology to the things I really loved, like music and literature.

So, what do you want to code? Websites? Games? iPhone apps? A startup that makes you rich? Interactive art? Do you want to be able to impress your boss or automate a tedious task so you can spend more time looking at otter pictures? Perhaps you simply want to be more employable, add a buzzword to your resume, or fulfill the requirements of your educational program. All of these are worthy goals. Make sure you know which one is yours, and study accordingly.

There's nothing mystical about it

Coding is a skill like any other. Like language learning, there's grammar and vocabulary to acquire. Like math, there are processes to work through specific types of problems. Like all kinds of craftsmanship and art-making, there are techniques and tools and best practices that people have developed over time, specialized to different tasks, that you're free to use or modify or discard.

Joel Spolsky posits that there is a bright line between people with the True Mind of a Programmer and everyone else, who are lacking the intellectual capacity needed to succeed in the field. That bright line consists, according to him, of pointers and recursion.

I learned about pointers and recursion in school, and when I understood them, it was a delightful jolt to my brain — the kind of intellectual pleasure that made me want to study computer science in the first place. But, outside of classroom exercises, the number of times I've had to be familiar with either concept to get things done has been relatively small. And when helping others learn, over and over

again, I've watched people complete interesting and rewarding projects without knowing anything about either one.

There's no point in being intimidated or wondering if you're Smart Enough. Sure, the more complex and esoteric your task, the higher the level of mastery you will need to complete it. But this is true in absolutely every other field. Unless you're planning to make your living entirely by your code, chances are you don't have to be a recursion-understanding genius to make the thing you want to make.

It never works the first time

And probably won't the second or third time

When you first start learning to code, you'll very quickly run up against this particular experience: you think you've set up everything the way you're supposed to, you've checked and re-checked it, and it still doesn't work. You don't have a clue where to begin trying to fix it, and the error message (if you're lucky enough to have one at all) might as well say "fuck you." You might be tempted to give up at this point, thinking that you'll never figure it out, that you're not cut out for this. I had that feeling the first time I tried to write a program in C++, ran it, and got only the words "segmentation fault" for my trouble.

But this experience is so common for programmers of all skill levels that it says absolutely nothing about your intelligence, tech-savviness, or suitability for the coding life. It will happen to you as a beginner, but it will also happen to you as an experienced programmer. The main difference will be in how you respond to it.

I've found that a big difference between new coders and experienced coders is faith: faith that things are going wrong for a logical and discoverable reason, faith that problems are fixable, faith that there is a way to accomplish the goal. The path from "not working" to "working" might not be obvious, but with patience you can usually find it.

Someone will always tell you you're doing it wrong

Braces should go on the next line. Braces should go on the same line. Use tabs to indent. But tabs are evil. You should use stored procedures, but actually you shouldn't use them. You should always comment your code. But good code doesn't need comments.

There are almost always many different approaches to a particular problem, with no single "right way." A lot of programmers get very good at advocating for their preferred way, but that doesn't mean it's the One True Path. Going head-to-head with people telling me I was wrong, and trying to figure out if they were right, was one of the more stressful aspects of my early career.

If you're coding in a team with other people, someone will almost certainly take issue with something that you're doing. Sometimes they'll be absolutely correct, and it's always worth investigating to see whether you are, in fact, Doing It Wrong. But sometimes they will be full of shit, or re-enacting an ancient and meaningless dispute where it would be best to just follow a style guide and forget about it.

On the other hand, if you're the kind of person who enjoys ancient but meaningless disputes (grammar nerds, I'm looking at you), you've come to the right place.

Someone will always tell you you're not a real coder

HTML isn't real coding. If you don't use vi, you're not really serious. Real programmers know C. Real coders don't do Windows. Some people will never be able to learn it. You shouldn't learn to code. You're not a computer programmer (but I am).

"Coding" means a lot of different things to a lot of different people, and it looks different now from how it used to. And, funnily enough, the tools and packages and frameworks that make it faster and easier for newcomers or even trained developers to build things are most likely to be tarred with the "not for REAL coders" brush.

Behind all this is the fear that if "anyone" can call themselves a programmer, the title will become meaningless. But I think that this gatekeeping is destructive.

Use the tools that make it easiest to build the things you want to build. If that means your game was made in Stencyl or GameMaker rather than written from scratch, that's fine. If your first foray into coding is HTML or Excel macros, that's fine. Work with something you feel you can stick with.

As you get more comfortable, you'll naturally start to find those tools limiting rather than helpful and look for more powerful ones. But most of the time, few people will ever even look at your code or even ask what you used — it's what you make with it that counts.

Worrying about "geek cred" will slowly kill you

See above. I used to worry a lot, especially in school, about whether I was identifying myself as "not a real geek" (and therefore less worthy of inclusion in tech communities) through my clothing, my presentation, my choice of reading material and even my software customization choices. It was a terrible waste of energy and I became a lot more functional after I made the decision to let it all go.

You need to internalize this: your ability to get good at coding has nothing to do with how well you fit into the various geek subcultures. This goes double if you know deep down that you'll never quite fit. The energy you spend proving yourself should be going into making things instead. And, if you're an indisputable geek with cred leaking from your eye sockets, keep this in mind for when you're evaluating someone else's cred level. It may not mean what you think it does.

Sticking with it is more important than the method

There's no shortage of articles about the "right" or "best" way to learn how to code, and there are lots of potential approaches. You can learn the concepts from a book [pine.fm/LearnToProgram] or by completing interactive exercises [codecademy.com] or by debugging things that others have written [learnpythonthehardway.org]. And, of course, there are lots of languages you might choose as your first to learn, with advocates for each.

A common complaint with "teach yourself to code" programs and workshops is that you'll breeze happily through the beginner material and then hit a steep curve where things get more difficult very quickly. You know how to print some lines of text on a page but have no idea where to start working on a "real," useful project. You might feel like you were just following directions without really understanding, and blame the learning materials.

When you get to this stage, most of the tutorials and online resources available to you are much less useful because they assume you're already an experienced and comfortable programmer. The difficulty is further compounded by the fact that "you don't know what you don't know." Even trying to figure out what to learn next is a puzzle in itself.

You'll hit this wall no matter what "learn to code" program you follow, and the only way to get past it is to persevere. This means you keep trying new things, learning more information, and figuring out, piece by piece, how to build your project. You're a lot more likely to find success in the end if you have a clear idea of why you're learning to code in the first place.

If you keep putting bricks on top of each other, it might take a long time but eventually you'll have a wall. This is where that faith I mentioned earlier comes in handy. If you believe that with time and patience you can figure the whole coding thing out, in time you almost certainly will. ■

Cecily Carver is a Toronto-based software developer for Bento Box Projects and co-director of the Toronto organization Dames Making Games, which aims to support women interested in making, playing, and changing. She leads programming and game-making workshops for beginners, and has spoken about her work at IndieCade, GRAND, and FanExpo.

Reprinted with permission of the original author.
First appeared in *hn.my/learn* (medium.com)

Ethiopian Binary Math

By JOHN H. LIENHARD

This is part of the long-running series, The Engines of Our Ingenuity, heard nationally on Public Radio.

THE SCENE IS a remote Ethiopian village in 1940. A Farmer offers his herd of 34 goats for sale. One goat is worth, say $7. The villagers don't know how to multiply, so they call in a shaman. They ask him to set a fair price for the whole herd.

The shaman digs two rows of small holes in the hard dry earth. He reaches into his sack of pebbles and goes to work. He puts 34 stones in the first hole on the left — one for each goat. He puts half that, or 17, in the next — half 17, or 8, in the next — and so on. He keeps dividing by two and dropping the remainder, until the sixth hole has only one stone in it.

Now he goes to the other row. He puts 7 stones — the value of one goat — in the first hole. He puts twice that, or 14 stones in the next hole, and so on. Now his deliberations begin.

He goes down the left-hand side, seeing whether the holes are good or evil. An even number of stones makes the hole evil. An odd number makes it good. Two holes are good. The holes next to them, in the right row, contain 14 stones and 224 stones. He adds those numbers together. The result is the fair market value of the herd. It's $238.

You and I know about multiplication. So we multiply the number of sheep, by the value of a sheep — 7 times 34. When we do that, we get $238. But that's just what the shaman got! So what in the world was all the business with the holes? And would he get the right answer with different numbers?

We try it with other numbers. It works every time. So we turn to a mathematician. He says it's not at all obvious. He puzzles for a long time. Finally he sees it. This Ethiopian shaman has created a remarkable algorithm.

All that business with the holes identifies the numbers in their binary form. That lets the shaman reduce multiplication to simple addition. He's multiplied just the way a digital computer does. Where did his method come from? How long have his forbears carried this rote tradition?

An anonymous genius lurks somewhere in the haze of his history. So we look at our own multiplication and realize that we too use ritual to find what 7 times 34 is. It makes no more sense to most people who use it than the shaman's holes. Our multiplication algorithm was also given us by an anonymous genius. He is also lost in rote tradition.

So how do we and that Ethiopian shaman differ? Very little, I reckon. Very little indeed. Of course, I wouldn't be surprised if he makes fewer mistakes than we do. ■

****

- Currie, W.S., Binary in the Stone Age. Geophysics: The Leading Edge of Exploration, March, 1985, pp. 50-52.

- The shaman's multiplication of 7 x 34:

```
row #1              row #2    the calculation

34 stones (evil)         7    evil   0
17        (good)        14    good  14
8         (evil)        28    evil   0
4         (evil)        56    evil   0
2         (evil)       112    evil   0
1         (good)       224    good 224
                                 _____
                                   238  = 7 x 34
```


John H. Lienhard is a Professor Emeritus of Mechanical Engineering at University of Houston.

Reprinted with permission of the original author.
First appeared in *hn.my/ethiopian* (uh.edu)

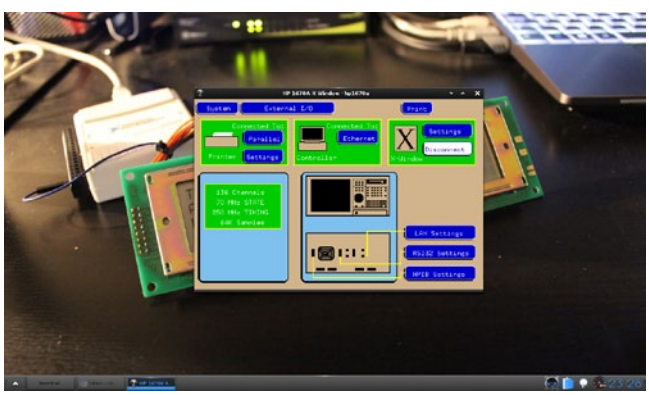
A Testament to X11 Backwards Compatibility

By ANDREW ROSSIGNOL

I RECENTLY SCORED A Hewlett Packard 1670A Deep Memory Logic Analyzer, and I finally had a chance to fire it up. This unit dates back to 1992 and is packed with all sorts of interesting options for connecting peripherals to it. One particular feature that caught my eye was the option to connect to an X Server.

HP 1670A Logic Analyzer

Here is the interface of the logic analyzer running on a remote X connection. I enjoy the color scheme.



HP 1670A user interface over an X connection :]

I will give you a quick explanation as to how I was able to set this up by modifying a couple of configuration files to enable remote X connections.

I run Linux Mint 15 with the e17 window manager (absolutely fantastic) and the gnome desktop manager (gdm). The first step was to assign my new logic analyzer an IP address as it does not support DHCP. This was fairly trivial, I merely assigned it a vacant IP on my network.

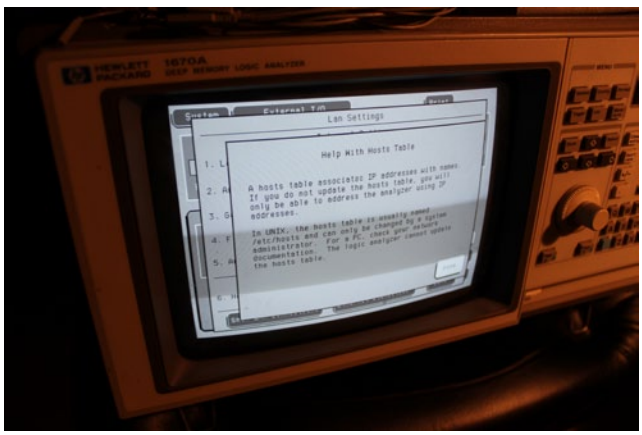
Here is the configuration menu of the logic analyzer sporting classic interface design complete with the X logo. Take note of then convenient arrows to indicate which port each button adjusts settings for.

Configuration Menu

I especially enjoy the rotary encoder to the right of the screen as an input device. It is quite tactile and is a fun way to input the IP address. All that it is missing is the ability to depress it.

IP Address Information, Boring.

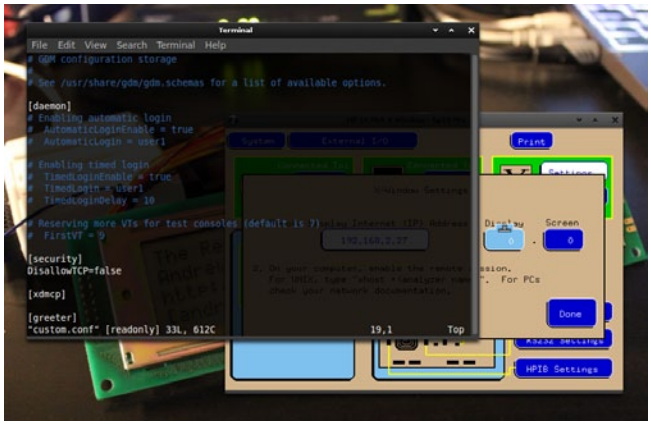
I also found some bonus help material about the hosts file on UNIX systems. I see everything has been status quo since 1992.



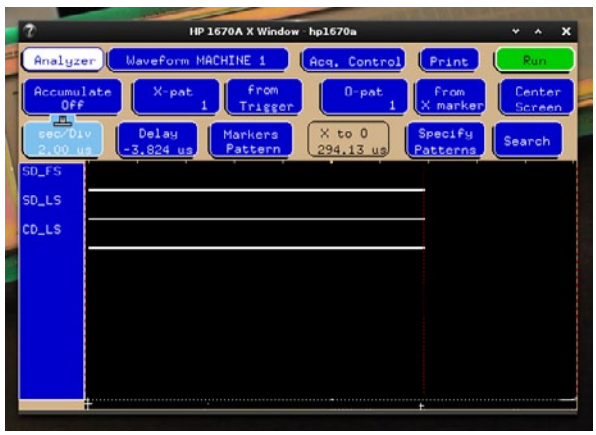
Hosts File information

Next, I had to make a couple of changes to configuration files to allow remote X TCP connections. I followed instructions from a question on *serverfault.com* to make this happen.

First, I modified `/etc/gdm/custom.conf` to allow `DisallowTCP = false`



I also modified `/etc/X11/xinit/xserverrc` and removed the "`-nolisten tcp`" option. A quick restart of gdm later and I was able to establish the connection between the logic analyzer and my laptop. I find this to be a rather interesting feature of this piece of test gear. It's a shame that more devices don't implement the protocol, this is quite a cool feature if you ask me.



Waveform Viewer

This all reminds me very much of the Chain of Fools video [hn.my/chain] from back in 2011 where Andy successfully upgraded from Microsoft's DOS 5.0 through to Windows 7 and was still able to play Doom and Monkey Island. I can definitely say that this is an impressive feat for a systems design house such as Microsoft, but the *nix's deserve some credit, too! ■

Andrew Rossignol is passionate about electronics and technology. He has been doing tear downs and coming up with fun technical projects since he was a little kid. Not much has changed since then!

Reprinted with permission of the original author.
First appeared in *hn.my/x11* (theresistornetwork.com)

How to Launder Bitcoins Perfectly

By OLEG ANDREEV

People often talk about privacy problems with Bitcoin: all transactions are public and every move is watched by millions of eyes. Where there's a problem, there's a solution.

Let's first define the problem more rigorously. There are two situations (ok, three) when you want to launder your coins.

First: You receive monthly salary on a single address and then want to do regular purchases with it. When you're buying a cup of coffee, the shop owner will see how much money you have, which might be unsafe.

Second: You want to buy something expensive, so you have to combine "change" from various addresses in a single transaction. This may link many of your private payment histories in one. Someone may connect the dots and make a full profile of a single person: what he eats, where he travels and so on. It's being done with credit cards already and people don't seem to like it very much.

Third: You sold something anonymously and your payment is being watched. If you later spend that money in the open, your identity may be revealed.

Bonus track: Some people think that "money laundering" is not sinful enough, so they invented "structuring laws," which forbid not only buying bad things, but also hiding the monetary trails even if you don't do anything illegal at all. If your method to launder bitcoins is screaming "LAUNDERING" on the blockchain (like with Zerocoin, using shared addresses or CoinJoin transactions), it's not good for you. You may get your privacy, but you also go to jail for "structuring." To be a law-abiding citizen you should not hide your financial history. The rest of this article is for pure entertainment only.

To address all of these issues we need to disperse and mix the funds in a way that their source or destination becomes statistically indistinguishable from any ordinary transaction.

You might do that with these ingredients: discover, insurance, split and swap.

Disclaimer: This is not advice, it's a technological overview for all those who are interested in privacy aspects of Bitcoin. Anyone can implement this or come up with an even better idea. This is not even my original idea. I recommend governments to shut down the entire network to prevent people from doing nasty things with Bitcoin. At the same time, there's an opportunity to use this scheme by undercover FBI agents to detect anyone mixing their bitcoins. Dear reader, please obey the laws and be a good, socially responsible person.

1 Your wallet app discovers random nodes on the P2P network (other instances of the same app) and posts a request to launder some bitcoins. When two wallets meet with similarly sized requests, they exchange information about some of the available coins. Each of them does statistical analysis of those coins and decides if the coin is "good enough." For instance, if this coin's history correlates as little as possible with the histories of the coins already owned.

2 When both nodes like each other's coins, they enter an insurance contract. Each party locks up an equal amount of coins in a single special transaction where coins can only be unlocked atomically and by mutual agreement. At the same time, each party can destroy both deposits (e.g., in case of timeout or misbehavior of another node). The amount of each deposit should be 200-300% of the amount to be exchanged.

3 Each node splits their coin in two parts. One part is to be exchanged now, another part is to be exchanged with some other node later. Parts of the coins should be equal. (This produces some correlation detectable on blockchain, but that's easy to fix with multiple independent transactions instead of just one.)

4 Each node tells another one an address on which to send a part of the coin. Each of them does that transaction. All the other nodes don't know about this swap of coins and therefore cannot link them together. If your coin was "tainted" (watched by adversary), half of it anonymously goes to someone else and in return you get some absolutely different coin. Insurance contract prevents a node from receiving a payment, but not making a payment back. Since there is no human supervision, anyone trying to cheat the scheme will get punished by an automatic destruction of his deposit (which is worth much more than just received money).

During one session (one insurance contract), nodes can swap more coins until they run out of coins or cannot provide each other with a statistically good one. When the session is over, insurance deposits are unlocked and nodes go talk to other nodes.

Think about it this way: you split all your money in 1000 pieces and send them to 1000 different random strangers via regular, statistically innocent transactions. In return you get 1000 pieces from all around the world that are not connected to each other in any meaningful way. 10 rounds splits money into 1024 portions, 20 rounds into over a million. In a short period of time you never expose more than a fraction of your funds and never receive more than a fraction of someone else's history.

How does this address our examples?

When you receive a monthly salary payment, you mix it with 1000 random users and in return get 1000 smaller pieces. It's like exchanging one $1000 bill for a thousand $1 bills. Then, you can go buy your coffee and no one will know how much money you have.

When you need to spend a lot of money at once, you do the same: take all your small coins, swap anonymously for other small coins and make a single payment. Your individual spending histories will be dispersed among thousands of random people. And the recipient of your payment will link together totally uncorrelated histories having nothing to do with you personally.

Finally, if some of your money is being watched ("tainted"), it will be moved to someone else completely. You yourself have little risk of getting someone else's tainted history because you never get more than 0.1% of it due to multiple rounds of splitting.

The UI for this can be quite simple. You install a special kind of wallet, load it with bitcoins, connect to the internet and click "Mix coins." The next morning all your coins are perfectly mixed with thousands of random strangers.

Again, this is not a ready solution, but a theoretical possibility for those who are interested in solving puzzles. Don't use this if the law forbids it. The law is very important. ■

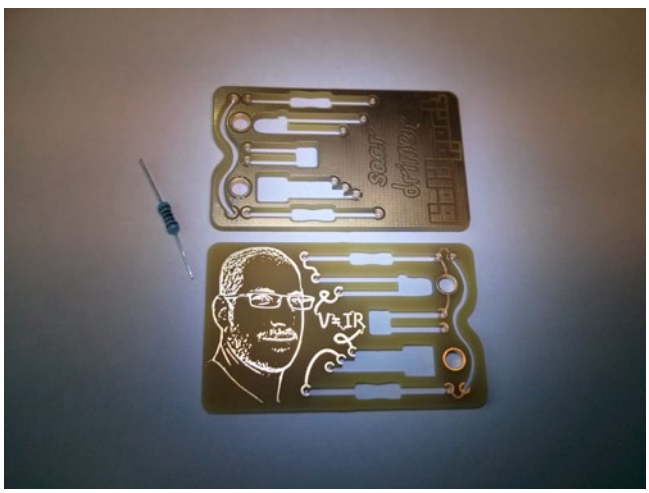
Oleg Andreev is a software developer from St. Petersburg, Russia interested in a variety of areas from user interfaces to networking protocols and cryptography.

Reprinted with permission of the original author.
First appeared in *hn.my/launder* (oleganza.com)

An Engineer's Emergency Kit Business Card

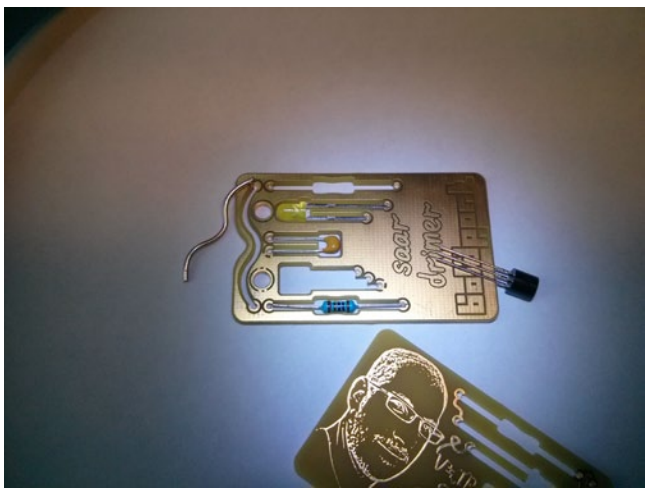
By SAAR DRIMER

Circuit board business cards have been done. But since circuit boards are, literally, my business [boldport.com], I felt that I needed one, too. Of course it also had to be special. Research and experimentation took a long time with this one, and the design even sat dormant, ready, for a while before I sent it out to fab.



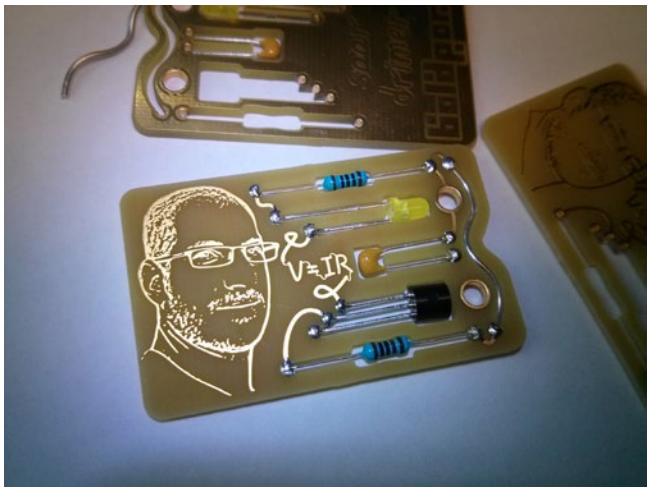
Without components

The concept was to have through-hole components embedded within the PCB and soldered lying down. The components — two resistors, LED, NPN MOSFET, and a capacitor — form a complete circuit so that when voltage is applied, the LED turns on.



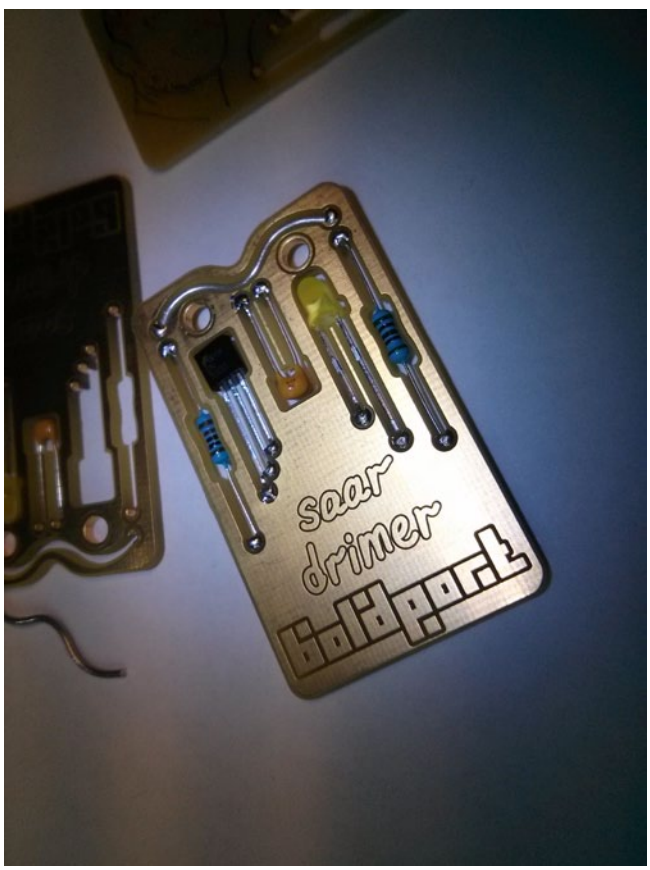
Sizing up the components. Notice the wiggly piece of solder that fits into one of the slots.

It's meant to be an engineer's emergency kit. When all hope is lost, the MacGuyver engineer could snap out one of the components and save the day. Recall the countless times you desperately needed a 1 KOhm resistor to fix an amplifier at a party, only to see the girl you were trying to impress slip away with an OCaml programmer? Never again with this little kit. You even have 2cm of solder in there to make sure the connection's electrically solid!



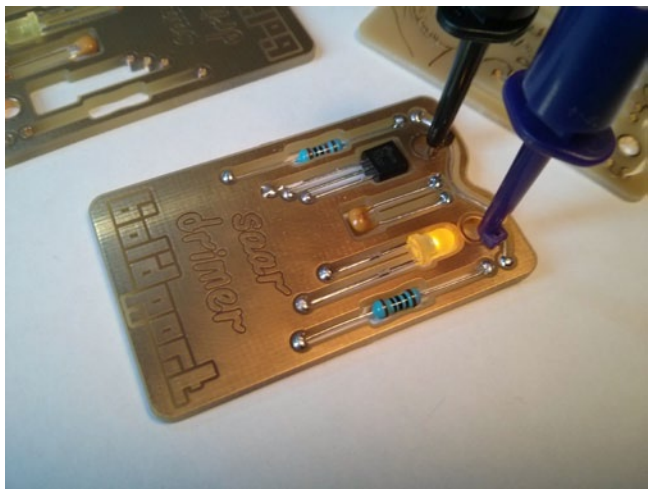
Components soldered into place (top side)

Consider the times when you were too drunk to recall Ohm's Law, yet was called in to fix a spaceship's control system. V=IR is written on the board to rescue you into awesomeness in spite of your inebriated state.



Components soldered into place (bottom side)

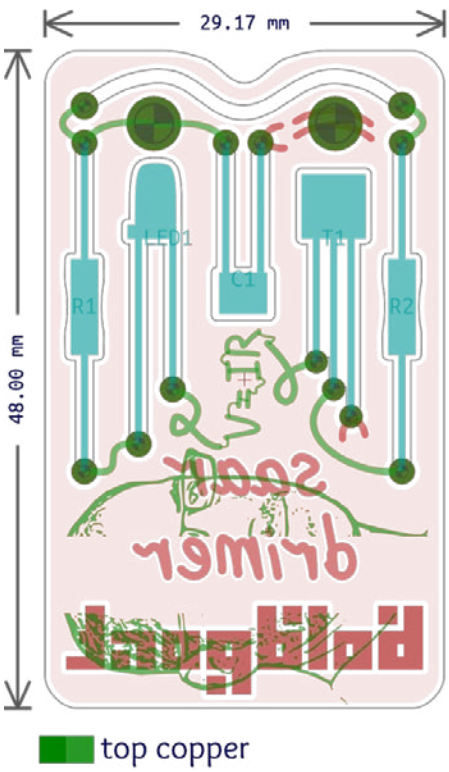
For those extreme situations when you need a Winston Wolfe, my details are there so you know who to contact when the going gets tough. Finally, as motivation, my disapproving mug is there to stare at you as you're going about your engineering super hero day.



It's a functional circuit! The LED lights up when you apply power.

The board was manufactured by PCB-POOL, without soldermask or silkscreen and using their default ENIG finish. This was the first PCBmodE board I've made with this fab, and they've done a great job. I particularly like that they send pictures of the board during the manufacturing process.

Banana for scale for us Reddit types



Manufacturing detail:

Unit of length: Millimetre (mm)
Number of layers: 2 ('top', 'bottom')
Thickness: 1.6 mm
Material: FR4, 1 oz / 35/35 um copper
Silkscreen: none
Soldermask: none
Surface finish: Any lead-free

Design information:

Board name: 'breakout'
Revision: A
Description: An engineer's emergency kit and business card
License: Apache Version 2.0
Company: Boldport Limited
Designer: saar drimer
Email: saardrimer@gmail.com

This open source PCB design was created using 'PCBmodE',
an open source software
http://bitbucket.org/boldport/pcbmode
http://boldport.blogspot.com
http://www.boldport.com

A view from Inkscape/PCBmodE. The assembly layer was used to size the cutouts. (That break in my face is an artefact from Inkscape's bitmap export)

Now I only need to figure out how to manufacture this design cheaply enough so I can actually give those kits away. ■

Saar Drimer is an engineer.

Reprinted with permission of the original author.
First appeared in *hn.my/businesscard* (blogspot.co.uk)

mailjet

EMAIL FOR YOUR APPS

SEND. TRACK. DELIVER.

Your one stop shop for **ALL your email needs.**
Manage lists as well. No extra fees for **Newsletters.**
Priority headers to deliver notifications in **real time.**

Go for mailjet.com