Interview: Colin Hayhurst on running a PaaS startup
StackBlaze’s co-founder shares his experiences

PaaS: the cloud on-ramp for PHP developers
AppFog’s Lucas Carlson on cloud deployment

Cryptography in PHP
Zend’s Enrico Zimuel on securing your data
Introducing the difficult second issue

It’s hard to believe that it’s already been a month since the first issue of Web & PHP Magazine. We debuted on the last Friday of March and, despite a few technical hiccups, got a great response from the community. Thanks to some handy re-tweets and a link from the venerable Chris Cornutt over at PHPDeveloper.org, we’ve racked up over 1,000 free downloads. Given the great content in this second issue, there’s no reason why it shouldn’t be just as popular.

First up, we’ve got our regular interview slot, filled this month by Colin Hayhurst from StackBlaze. We bumped into Colin at a PHP London meet-up back in February and were intrigued to hear about how StackBlaze is making simple, clean Platform as a Service (PaaS) deployment a reality for PHP developers. To stir up a bit of competition, we’ve also got a short feature from Lucas Carlson at cloud rival AppFog, running through some of the basic principles behind the PaaS movement. Filling out the front of the magazine is a preview of next month’s PHP Summit event by yours truly and a short column on architecture issues by thePHP.cc’s Stefan Priebsch, while the closing pages are devoted to a real in-depth look at cryptography from Zend’s Enrico Zimuel.

As ever, if you’ve got something that you’d like to see us cover, or an article of your own that you’d like to submit, feel free to get in touch, either on Twitter (@webandphp) or via email (louish@websmedia.com). While we find ourselves stumbling into themes now and then (databases last issue, PaaS today), we like to keep things as open and wide-ranging as possible, and we’re hoping to introduce more community-focused columns and features over the next few months. As Zoë Slattery emphasised in our first issue, one of PHP’s greatest assets is the network of enthusiastic people across the globe who use it every day.

An infamous hatchet job currently doing the rounds describes the PHP ecosystem as a “community of amateurs”, claiming that “[v]ery few people designing it, working on it, or writing code in it seem to know what they’re doing.” For what it’s worth, we couldn’t agree less. PHP certainly has its newbies and its hobbyists, just like any other project, but it also boasts one of the most committed core development teams we’ve ever seen, a group of individuals volunteering their own time to constantly update and improve the language. When you throw in an active community of third-party developers and some of the most high-profile corporate users out there, you get a pretty potent cocktail of experience, skill and, most importantly, enthusiasm.

Louis Goddard
Editor

www.webandphp.com
Colin Hayhurst is the co-founder of StackBlaze, a PaaS startup based in the UK. We asked him what challenges face the would-be PHP entrepreneur in today’s risky economic climate, what he thinks of PHP 5.4, and why running a cloud outfit is one of the most difficult jobs around.

by Louis Goddard

You launched a PHP startup recently. What did it feel like to do that, and how have things been going since?

It’s been a fantastic experience and I’ve felt a whole lot more fulfilled than in any previous job. For me, you can’t beat working for yourself with other founders. It’s not that I’m a control freak, indeed many would say I’m the very opposite of that. It’s just that it sucks in so many ways to be slaving away being part of somebody else’s vision.

It’s been quite a climb to get where we are now, with our recent commercial soft launch of StackBlaze. Indeed nine months have elapsed since we started on the product development. My own involvement has not been as intense, until recently, as that of my co-founder who has handled almost all of the technical development. So the length of that time could have been more frustrating, but there was plenty going on in the meanwhile. Apart from all the commercial aspects of launching, we had a private alpha and private beta programmes in that period. I also worked part-time in another business, where my job was helping a portfolio of technology startups, until September 2011. We are really just starting on our adventure so we can expect lots of challenges and fun ahead.

How hard is it to start a company in the current economic climate? How did you get funded?

The current economic climate makes it easier, in my opinion. Most ambitious startups are looking to disrupt current businesses, business models and markets, so in a downturn they have an advantage over established players. Startups have the advantage of being nimble, having lower costs and no legacy to hold them back. Perhaps raising investment money is a little harder, but I don’t think it’s that much different for startups than in the past. Yes, it’s very tough if you are looking for capital in an established business, but for startups there have always been very few sources of good finance willing to take the necessary risks.

We were lucky enough to raise a small grant from the UK’s Technology Strategy Board. That was dependent on finding some match funding, in cash, which I was committed to doing. Since that grant expired I have been funding the business myself with proceeds from a previous venture. Now we have reached the point where we can bring in revenues from the product.

Can you tell us a bit about your background?

In 2005 the software business I ran, Century Dynamics, was sold to a NASDAQ company, whereupon I took a swift exit. As a co-founder at Century Dynamics, I went on the whole roller-coaster journey, from bootstrapped start to trade exit. Doing a startup is an intense experience and frankly addictive, so I have missed that for six years. However, I deliberately avoided doing another startup in 2005, because I wanted to see more of my kids. Plus get to sleep at night, a whole lot better.

Century Dynamics developed and sold high-performance computing software. Indeed, our first notable achievement was taking algorithms and software technology run on Cray supercomputers and providing it as a commercial product on PCs in the mid-to-late
‘80s. We even had an interactive graphical user interface from the start; on an IBM PC, that made us pretty radical. Indeed at that time and for a few years, many prospective customers thought we were crazy. But gradually, over time, and with persistence we came to be the worldwide industry standard, in our niche. Later we took that expertise and technology into other areas and established new products and markets in the space, oil and automotive industries. Luckily for me along the way I got to work on some incredible projects, most notably the International Space Station and the CERN Large Hadron Collider.

I actually studied and trained as a Chartered Engineer and started my career as a design engineer in manufacturing engineering. But since software had been my main interest as a young teenager I chose deliberately to get more involved in software opportunities in engineering. Because FORTRAN was, and remains, the best language for high-performance scientific computing, that became my main language of choice until I stepped back from full-time programming as I focussed on running four product teams and growing a company.

From 2005 to 2011, I was helping a large number of mostly software startups with a variety of things like investment readiness, mentoring, business models and recruitment.

How did you meet your co-founder, James Cunningham?

We met a fantastic hacking event called Young Rewired State, where James was presenting to a distinguished panel of judges: he actually won the top prize presented by the editor of Wired. I was there to see my eldest son present, so it turned out to be a lucky coincidence that we met. We kept in touch and, along the way, discovered that we were both looking to do a software startup. This led to discussions about the ideas behind StackBlaze and we eventually decided to start a business in May 2011, after James finished his college year. I’ve worked with a good number of very talented developers over the years, some of them very young, so the experience has not really been anything new. Mind you, James is unique, in my experience, in having both an extraordinary technical talent and a good business head, for one so young.

Why did you choose to launch a PHP startup?

Well, the choice was not so much the language but the problem we were looking to solve: how to provide faster, cheaper and better web hosting. James had a lot of experience with hosting games servers and scalable applications: among other things, he developed and self-hosted the fastest-growing Twitter application ever, serving over 3 million new users in one day.

The fact that StackBlaze is focused on PHP applications is because James’s core skills and experience are in that space. But it is also because the PHP market is not as well-served by modern hosting platforms. Platforms such as Heroku have tackled a similar space for Ruby and they provide a lot of value to developers, in that sector, looking to host their applications. Indeed, many would argue that Heroku has contributed significantly to the popularity and growth of Ruby. So it would be great if an innovative platform for PHP helped in a similar way.

What are your impressions of PHP compared with other languages? Is PHP 5.4 a solid bet for mission-critical, future-proof apps?

One of PHP’s biggest assets is also its biggest failing when compared to other languages. That’s its backwards compatibility, which has led to inconsistent parameter-ordering, a bloated core library and the like. But this is also where it excels. You can go ahead and run an older application, or update a PHP version, feeling confident that the application will work.

While PHP 5.4 does contain a lot of changes, and drops support for a lot of older language features, this is not a sudden change. Some of the features being dropped have been marked as deprecated for the last five years, ensuring that any application from that period should work without a hitch: does anybody actually use ASP style tags any more? As well as this, PHP 5.4 appears to have taken inspiration from other, more fashionable languages like Ruby and Python of late, including things like short array syntax.

What does StackBlaze do? What problem are you trying to solve?

Fundamentally we want to make hosting a no-hassle issue. Why should hosting require specialist knowledge? Nowadays we expect mobile applications to be intuitive, so we should be aiming for hosting to be like that too.

As far as possible, scalability should simply be taken care of and not require technical knowledge. Developers and others deploying websites should not have to worry about their application crashing or slowing down due to a surge or a spike in traffic. High traffic-growth is what we all hope for and yet, so often, it turns out to be a problem.

So one specific problem we are looking to solve is the situation where your application or website is featured by a prominent blog. You get a surge in traffic and the website goes down. How tragic and ridiculous is that? Having a hosting platform which is truly elastic and which can automatically load-balance traffic across multiple servers would solve this problem. That’s exactly what StackBlaze does.

Often the scalability challenge is tackled by using dedicated servers, but that’s an expensive solution, both in money and often in time. With StackBlaze we’re taking a different approach, aiming to provide...
the performance of dedicated servers but at a price-level closer to that of shared servers.

The way StackBlaze works is that your application runs on multiple servers and is automatically load-balanced across these servers based on the resources needed by your application and others. StackBlaze leases dedicated servers from one of Europe’s biggest data centres and we then use our own grid network technology to load-balance demand so that your application gets the resources it needs.

StackBlaze requires little specialist knowledge, and you certainly don’t need to rely on a systems administrator to sort out your hosting.

How can you compete with the plethora of hosting solutions out there, particularly more established PaaS solutions like AppFog?

Clearly we are not competing directly with shared nor dedicated hosting providers since we provide the performance of the latter at the price of the former. Equally, we are not competing directly with the large providers such as Amazon and Rackspace, who require more specialist knowledge and a deeper pocket. So our direct competitors are recent startups like ourselves, such as AppFog or PagodaBox. What differentiates StackBlaze from these companies is that our service is truly elastic with load-balancing being completely automatic. Also, since competitor services are invariably built on top of Amazon Web Services, they must charge a premium on top of the costs they pay to Amazon. Since StackBlaze runs directly on top of better price/performance data centre servers, we can pass this advantage on to our customers.

How can you compete with the plethora of hosting solutions out there, particularly more established PaaS solutions like AppFog?

Clearly we are not competing directly with shared nor dedicated hosting providers since we provide the performance of the latter at the price of the former. Equally, we are not competing directly with the large providers such as Amazon and Rackspace, who require more specialist knowledge and a deeper pocket. So our direct competitors are recent startups like ourselves, such as AppFog or PagodaBox. What differentiates StackBlaze from these companies is that our service is truly elastic with load-balancing being completely automatic. Also, since competitor services are invariably built on top of Amazon Web Services, they must charge a premium on top of the costs they pay to Amazon. Since StackBlaze runs directly on top of better price/performance data centre servers, we can pass this advantage on to our customers.

What can we expect to see in StackBlaze in the future?

We went live publicly at the start of March and were pleased to quickly get a good set of initial customers who have provided some great feedback. As a result of that feedback, we have improved the user interface to make set-up even more straightforward and rolled that out over Easter. We still see many ways that we can make the whole experience more intuitive, so we will continue to work on that.

Other high-priority items which we expect to roll out in the next few months include Git syncing and ssh access. We’ve also started including one-click install of popular applications like WordPress, phpBB, Mediawiki and Drupal, but we fully expect to add more of these.

You’ll see some other things, for sure, but we are keeping an open mind on the priorities because we want those to be primarily driven by users’ needs. Indeed if you are reading this and would like to see something supported in StackBlaze, please let us know.

What is your biggest challenge moving forward?

Making people aware that your product exists and communicating why it offers distinct advantages over other solutions is a challenge for all businesses, of course. While the potential market for PHP hosting is very large, the big issue we see is that it is also very diverse: at the extremes, you have everything from small WordPress blogs to huge applications like Facebook. Then you have other segments like e-commerce sites and web agencies. So our biggest challenge lies initially in establishing a well-known presence and great reputation in one or more of these market segments.

What have you made? Have you had any major crises, and are you living up to your promise of 99.95% availability?

The big mistake we made when we started the company was having two projects. We originally built a social web application, which used StackBlaze, and which we thought could be a product in its own right. It may well have been, but we should have concentrated on one project from the start. Dropping that project, last July, caused a lot of heartache at the time, but things improved dramatically when we decided to concentrate on StackBlaze.

So far since public launch we are at 100% availability, but that’s only a relatively short time span. Actually, we have been running private alpha and beta programmes since August 2011 – in that time we had one outage for a RAID failure in December, which was fixed by the data centre within one hour, so our availability has been 99.98%.

What are your six top tips for somebody thinking of launching a startup?

1. Orient your startup around something you are passionate about.
2. Build something people actually need or will realise they need.
3. Get feedback and listen to customers and prospects.
4. Work with co-founders you can trust.
5. Expect an emotional roller-coaster.
6. If you can’t handle uncertainty, don’t do a startup at all.

Bio

Colin is a co-founder of StackBlaze, providers of truly elastic PHP web hosting. A software and startup addict at heart, he was previously co-founding CTO, then CEO of a software company, which successfully exited to a NASDAQ company.
Meet 3 of the most inspiring presenters in the PHP space for 3 days filled with tutorials:

- Sebastian Bergmann
  - Update PHP: Leverage New Features and Technologies Target
  - Unclean PHP: Identify, Refactor, Avoid
  - Testing PHP Applications: Fundamentals
  - Modern Version Control with Git
  - Continuous Integration with Jenkins

- Arne Blankerts
  - Best Practices: From the Real World for the Real World
  - Safely Prepared for Errors
  - Insider’s Tip XML: Applications the Smart Way
  - Tuning for Web Applications
  - It is best to expect the worst!
  - Attack! What makes attacks on the Web successful

- Stefan Priebsch
  - Object-Oriented Programming (OOP) in PHP I: Fundamentals
  - Object-Oriented Programming (OOP) in PHP II: Advanced Topics
  - Design Patterns I: The Most Important Standards
  - Design Patterns II: Integrating Sophisticated Patterns
  - Frameworks: The Basics in Three Hours
  - Why the Tower of Pisa is Leaning

More information: www.php-summit.co.uk
Three days of training for PHP developers of all skill levels

PHP Summit 2012 UK

Next month, the Regus Business Centre in Victoria will play host to three days of intensive PHP training, provided by the team from thePHP.cc – Arne Blankerts, Sebastian Bergmann and Stefan Priebsch. We talked to Mark Hazell, Software & Support Media’s Managing Director of International Operations, to find out more.

by Louis Goddard

We work to a hectic schedule here at S&S Media. As I write this, I’m sitting in a conference room in Victoria’s Park Plaza Hotel, lending a hand at this year’s JAX Days training event. The bacon and egg sandwiches are out, the orange juice is flowing and the coffee’s strong – just what’s needed after a 6 a.m. start and an hour or so on London’s decidedly un-Olympics-ready suburban rail system. I’m here for the full two days, covering a total of seven in-depth Java sessions from Ted Neward, Russel Winder, Ian Robinson, Adam Bien and Jim Webber. Yet in five short weeks, I’ll be back in Victoria for our next event – PHP Summit, three days of training for everyone from hardcore LAMP veterans to fresh-faced newbies making their first steps with PHP. It’s quite a feat of organisation, but somehow we manage to get it done.

While we’re all great fans of the Park Plaza (particularly their sandwiches), we’ll be switching to Regus’s Victoria Business Centre for PHP Summit – it’s just down the road in the landmark Portland House, offering much more space and a lighter, airier atmosphere. More importantly, it’s right next to Victoria Station, providing easy access to the District, Circle and Victoria underground lines, as well as mainline trains to South East England and an express train to Gatwick Airport. With any luck, the great weather that London has been blessed with for the past few weeks will continue, and attendees will be able to enjoy crisp mornings and unseasonably balmy evenings on Tuesday 8th, Wednesday 9th and Thursday 10th of May. West London certainly isn’t short of swanky watering holes at which to soak up the sun, and, for the pastorally-inclined, the beautiful Battersea Park is just across the river.

But enough of the surroundings. I spoke to S&S Media’s Managing Director of International Operations, Mark Hazell, for a run-down of what’s actually going on. So what’s behind PHP Summit? “PHP Summit is part of S&S Media’s Developer.Class programme,” explains Mark. “It sits somewhere between a full-blown conference and a simple one-day training session, offering a range of in-depth workshops from three different speakers over three days.” PHP Summit will be dominated by three big names in the worldwide PHP community: Sebastian Bergmann, Stefan Priebsch and Arne Blankerts. Together, Sebastian, Stefan and Arne form thePHP.cc, a PHP-focused consultancy firm based in Bavaria, and they work in three interlinked speciality areas.

As the creator of the popular PHPUnit tool, Sebastian is well known as an expert on unit testing and quality assurance. As he notes on his website, “[e]ven
experienced software developers need to expect the unexpected from time to time”, and that’s precisely the theme of his sessions at PHP Summit. “During his morning workshop on Wednesday 9th, Sebastian will walk attendees through the fundamentals of unit, database integration, edge-to-edge and end-to-end testing,” Mark tells me. “Later in the day, he will run a more advanced class on the same topic, providing a deep-dive into the more complex features of PHPUnit, with a particular focus on the challenges of applying robust testing procedures to legacy code-bases, a theme picked up from his Tuesday session on static analysis with ‘unclean’ legacy code.” Rounding out Sebastian’s group of sessions will be a couple of talks on the latest developments in PHP, including the new features of PHP 5.4 and distributed version control using Git, as well as a session on continuous integration with Jenkins.

While Sebastian is focused on pre-deployment testing, Arne Blankert’s passion is security – for a taste of his style, check out his column in our first issue, where he considers the security risks that can come from assuming too much:

“Our world is based on assumptions. It may not always be obvious, but the majority of decisions you and I make every day, the very core of our interactions, are almost always based on assumptions: assuming that you and I mean the same thing when naming it, assuming that a certain gesture will encourage the other or support a speech. You nod in agreement, shake your head to show disapproval. Or do you? Did you ever ask yourself, whether or not all those tiny things mean the same things in other parts of the world? What happens if they don’t?”

At PHP Summit, Arne will host six half-day sessions covering a variety of security topics, all with a practical bent. What exactly does he have in store? “Beginning on Tuesday, Arne will provide a down-to-earth talk on PHP best practices ‘from the real world, for the real world’, engaging with attendees through an audience-driven live-coding session,” says Mark. “This will be followed up by a workshop on error handling, a session on practical and secure usage of XML (JSON hasn’t achieved world domination quite yet) and a talk on tuning to avoid bottlenecks.” But for those looking for a real, practical security advice, Thursday will be the day to watch. In the morning slot, Arne will be presenting ‘It is best to expect the worst!’, a session for “security-aware developers who want to treat themselves to an upgrade of their professional paranoia.” As always, to protect effectively against the latest attacks and exploits, a good developer needs a fundamental understanding of how they work, and this talk will provide a high-level introduction to a number of the most popular methodologies. Finally, Arne’s contribution to the event will be rounded off by an afternoon session covering specific attack vectors and techniques, culminating in a live attack on a ‘black box’ demo application.

If Sebastian and Arne focus on practical problems, Stefan Priebsch is more cerebral, specialising in high-level architectural issues. Tuesday will be taken up entirely by a two-part primer on object-oriented programming. It’s a topic that’s been around for a while, but, according to Mark, “a lot of people still don’t fully understand the basic principles, particularly as they apply to PHP.” While the morning session will focus on fundamentals, Stefan will delve slightly deeper in the afternoon, covering topics such as dependency injection and abstract classes through a live-coding session. Wednesday will follow a similar two-act structure, with morning and afternoon sessions on design patterns, the emphasis being placed on tried-and-tested solutions – as Stefan asks, “Why reinvent the wheel every time?” The afternoon session will be of particular interest to advanced developers, including information on persistent objects in relational and NoSQL databases, temporal patterns and enterprise integration patterns. Stefan will use the Thursday to go into detail about some of the options provided by current PHP frameworks, including practical advice on their various advantages and disadvantages, before finishing off with ‘Why the Tower of Pisa is Leaning’, a look at the importance of non-functional aspects in technical decisions, with particular reference to current trends: distributed caching, message queues, NoSQL and others.

All in all, it looks like PHP Summit will provide a pretty serious three days of training, with enough variety to satisfy all tastes and ability levels. “It’s very much a pick-and-mix sort of event,” Mark clarifies – “attendees can tailor each day of talks to their specific needs and skill-sets.” Nine-to-five over three days works out to 24 solid hours of talks, workshops and networking – I’d challenge anyone to come away from that without learning something new about their language, shaking up their methodological assumptions and changing the way they work for the better. If you’re interested in attending, head over to the PHP Summit UK 2012 registration site now. Three-day tickets are currently available for £995, two-day tickets for £695 and single-day tickets for £395. For Web & PHP readers, we’re even offering a special 15% discount – simply enter the code ‘WAP12’ during registration. Whether you’re an old hand or a young gun, it promises to be a memorable few days – I look forward to seeing you there!
Your software is perfect. You don't need us.

http://thePHP.cc/get/there?
Mankind has been constructing buildings for over 5,000 years. The Great Pyramid of Giza, for example – undoubtedly a masterpiece of architecture and engineering – was built around 2600 BC. Constructing buildings has a few interesting basic conditions. First of all, there is – and always has been – a huge demand for buildings, driven by the rapidly growing world population. The requirements of a building are quite different depending on where it is located. In addition, environmental conditions are – or at least can be – constantly changing over time. Still, the most important parameters, like average family size and weather conditions tend to change rather slowly.

If you count in punch card-programmed looms, the concept of software is little more than 200 years old. Electronic digital computation is less than 100 years old. 30 years ago, the available main memory size of a computer would allow for creating software with a complexity that, compared to a building, would probably match a soggy cave, hopefully at least heated by a campfire.

Despite the enormous growth of the internet and the fact that computers today are ubiquitous and thus also account for a big demand for software, the lessons learned from a few decades of creating software certainly cannot match the lessons learned from a few millennia of planning and constructing buildings.

Not long ago, software was mostly stand-alone. Back then, data would mainly be keyed in, and reports printed out. Subsequently, data-exchange between computer systems became more and more important. Today, the focus is not only on exchanging data, but on keeping data synchronised between different systems.

Today, there are virtually no stand-alone systems any more. Software interconnects to and interacts with other computers. Buildings are no different. An architect needs to think about the environmental conditions, for instance. Are we building on sand? If you’ve read the Bible, you know how this will end. If you haven’t, take a look at the Leaning Tower of Pisa.

If we are sure that we have chosen the right place to construct our building, further questions arise. Where do energy and water supplies come from? Are we planning to heat the house with oil or gas? We probably won’t enjoy ourselves a lot in winter if we plan for gas heating when there’s no gas pipeline into our house. In warmer countries, central heating may not

**Why every developer needs to think about plumbing**

**Stefan Priebsch on ... seeing the bigger picture**

Software architecture is called ‘architecture’ for a reason. There are many things that a developer can learn from the history of physical buildings, all of which can help in constructing stable, reliable, scalable web applications.
be an issue at all, at least, as long as the climate does not change significantly.

Supply is not the only thing that matters, by the way: where does used wa-
ter go, for example? Creating our own soakaway might be a good idea at first
stance, but it has a limited capacity. Or we might pollute the drinking water in
our own well. We thus realise that we just cannot just build a house in some
remote location unless we are willing to generate our own energy, dig our
own well and hope that some means of mobile internet access is available.

While a self-supporting building might be quite appealing, as soon as tech-
ical problems occur, or we are short of wind, sun, or whatever we create
energy from, a large central energy supply and redundant distribution network
suddenly comes in very handy.

Software is far less tangible than buildings. Nobody will sue you for trying to
create a bloated, oversized or completely inappropriate architecture, even if
that could really help in some cases. But if you submit blueprints to your local
building authority, they will be put on public display for feedback. Even if all of
this seems like a big hassle, it is, in fact, a rather efficient sanity check.

Beyond all those technical issues there is another thing to keep in mind:
a building should fit nicely into its environment. You just cannot put a post-
modern skyscraper into a traditional Bavarian village. Well, in all honesty, you
could, but you would certainly have to expect a lot of trouble.

Software architects must be able to see the big picture. This includes being
aware of neighbours, suppliers, outlets, and, most importantly, taking impor-
tant non-functional requirements into account.

Stefan Priebsch unites expert knowledge with an extraordinary sense of when to use which tool.
His specialities are object-oriented development and software architecture. As an internation-
ally-acclaimed author and speaker he thrills auditoriums and likes to share his tremendous practical
experience.
Developers, especially PHP developers, are very pragmatic people. They create and invent things that change people’s lives. Look at Mark Zuckerberg, with Facebook written in PHP. Look at Jason Gendron. You don’t know Jason Gendron? Jason Gendron is just like you, a guy with an idea. He built a service that created communities out of Twitter friends. All of a sudden, he had tens of thousands of people using it and was spending half his time fixing and tuning what were supposed to be “managed servers”. He tried Platform as a Service (PaaS) and, all of a sudden, he could spend his time writing code and not managing servers. This let him have enough time to bootstrap into a profitable startup and quit his job. PaaS changed Jason’s life and it can change yours too.

As more applications move to the cloud, the complexity of application development has increased, as has the demand being placed on developers around managing and maintaining a growing number of applications. Developers are being pulled into configuring servers and load balancers and other infrastructure tasks. The introduction of PaaS technology moves many of these sysadmin tasks off developers’ backs and makes their lives easier.
First, let’s take a look at the traditional model for app development, deployment and management. In the traditional model, development requires many tedious tasks centred around allocation of machines, resource-assignment for the machines, specification of machines, clarification and excess communication related to all these things, some guessing about what the future holds and allocation of financial capital to buy the machines. Then comes the setup and configuration, then resource-placement of the machines in a data centre, co-location facility, or in some cases a closet inside the company’s building. These approaches were challenging at best – at worst, they were done by trial and error.

The Software Development Life Cycle (SDLC) in a traditional environment is frequently composed of hundreds of steps for each small vertical integration into the system. From top to bottom, each change has a ripple effect when the underlying systems change, and often even when the software changes just the slightest bit.

Environmental conflicts are one of the biggest ongoing problems of traditional environments. Because of the work and effort that goes into setting up an environment, there are frequently disparate environments for development: QA, test, UAT and production. Configuration changes cause conflicts, not to mention the host of problems that can arise around operating systems, event variables and server variables. Each environment requires slight modifications to work and, even worse, sometimes requires major modifications to run. These modifications then lead to the testing nightmare of inaccurately regressing every single environment every single time a bad deployment is made. This process creates massive ongoing operational and capital expenditures for a development shop.

These traditional environment issues have cost the development industry billions of dollars over the years. However, a revolution is brewing and already disrupting the way in which software development is done. Cloud-based PaaS does away with this traditional software development nightmare. Developers no longer need to focus on managing operations in order to get their work done. This move is as huge as the move from assembly to C/C++ or C/C++ to higher-level abstract languages like Java, C#, or Ruby. The move to PaaS, and the corresponding removal of the operating system barrier, enables vast improvements in the way software is developed.

With a PaaS environment replacing a traditional environment, a team or a single developer can go directly to building out a minimum viable product (MVP). This MVP can be deployed with every single build, providing continuous deployment and integration. This development practice leaves more time to add new features to the MVP and to do testing, gain user feedback, create features and more. The PaaS environment frees the developer from spending the notorious hours and days doing configuration, setup, and installation. With PaaS, you start building, push to your repository, view your creation live and you are up and running. Yes, there is still some setup. But instead of days, weeks, or even months to allocate systems, environments, tooling, and other systems, an environment can in many cases be set up and deployed in minutes. This environment can then be duplicated as needed. In many PaaS systems, the environment can even be stored as a schema or template, allowing easy recreation at a future time. Setup, one of the biggest operating system barriers, is practically non-existent within a PaaS environment.

The software development life cycle that is cumbersome and time-consuming can now be reduced to a mere tracking of the project. PaaS allows for greater time for team communication and product planning, lower overhead, and either faster iterations or lean continuous deployment. These things, while available to some degree in traditional environments, required countless additional hours to implement. With PaaS, most of these things are available immediately or with a simple SaaS solution used in conjunction with the PaaS offering. At the end of the day, PaaS makes the life cycle shorter, easier to adhere to and easier to maintain while making it easier to attain velocity with less inherent risk and fewer roadblocks.

The core principles of PaaS
Let’s take a look at a deployment process of a PaaS solution compared to a traditional environment. First, we’ll dive into the traditional deployment:
FeaturePaaS: the cloud on-ramp for developers

1. get a machine or instance to run the application on;
2. load the operating system;
3. set up the networking and prepare it for inclusion in the environment where it will be placed;
4. set up the hosting web server or service files and folders in preparation for deployment;
5. build/set up the application autonomously from the system itself, preferably on a development machine;
6. verify that the deployment of the configuration can be moved from one environment to the next;
7. push the code-base and application dependencies into source control;
8. in some way, get the application from source control to the server that was previously created: move, x-copy, use an .msi installer, bash script or deploy.

That’s eight steps, some of which are extremely time-consuming and painful. Let’s take a look at what we’d have to do to get the same exact application deployed to a PaaS solution (we’ll use phpfog.com in this case, but there are other platforms out there):
1. tell the PaaS the domain name you want;
2. push the code-base and application dependencies into source control; GitHub is always a good place to work from;
3. push your code live (see Figure 1) by clicking ‘Create’ or using a simple command line tool.

Open source
When looking to adopt a PaaS solution, you will want to make sure that you aren’t locked in to a solution that restricts a developer’s choice of frameworks, application infrastructure services and deployment clouds. One way to avoid vendor lock-in is to look at open source offerings, which will ensure that you will not be locked into a single framework, single set of application services or a single cloud.

One example of such a solution is Cloud Foundry. Cloud Foundry handles Application Life Cycle Management extremely well for many languages including PHP, Ruby, Node, Java, Python, Erlang and more. It is also open source on GitHub, which means that the open source community can add features, languages, technologies and services faster than any closed source option. This matters because your can run Cloud Foundry on your laptop for development and then push to production.

Once you combine a PaaS service (like phpfog.com or AppFog) with Cloud Foundry, you have the ability to abstract the PaaS even further and provide a vastly streamlined interface and command line. Even though AppFog will provide all of the Cloud Foundry functionality, the AppFog PaaS will extend the Cloud Foundry abilities further.

Conclusion
There is nothing that compares to the simplicity and velocity of building applications for PaaS deployment. The practices are already causing reverberations throughout the industry – companies are becoming faster, deploying faster and more frequently, and meeting customer demand more efficiently. It may not be magic, but it sure is easy!

Lucas Carlson is the CEO of AppFog, which he founded in September 2010. Before AppFog, Lucas led the development of MOG, a music-based Web 2.0 social networking application, with Ruby on Rails and MySQL. At MOG, he wrote the majority of the code base, created an architecture that could scale as the community grew, and hired, led and cultivated a technical team. MOG.com is currently one of the five most popular Ruby on Rails sites on the web. Lucas has authored over a dozen Ruby libraries, including the Ruby Cookbook, and contributed to various others including Rails and RedCloth. Lucas participates in industry groups and speaking engagements on a regular basis around the country.
Cryptography in PHP

How to protect sensitive data in PHP using cryptography

"The mantra of any good security engineer is: ‘Security is a not a product, but a process.’ It’s more than designing strong cryptography into a system; it’s designing the entire system such that all security measures, including cryptography, work together." – Bruce Schneier

by Enrico Zimuel

If you are a professional web developer, security is an important aspect of your job. If you manage critical data in your web application, like users’ passwords or credit card numbers, you should protect it. But how? Cryptography can be the answer, but you need to know how to use it and, most of the time, this is not so easy. In this article we will introduce the use of cryptography in PHP.

A brief introduction to cryptography

Cryptography (or ‘cryptology’, from the Greek ‘kryptos’ – “hidden, secret” and ‘graphein’ – “writing”) is the study of secret messages and the practice of how to protect information. Cryptography is a multidisciplinary science that involves mathematics, computer science, linguistics, software engineering and more. It protects information using special algorithms to transform a message into an unreadable format. Only the owner of the message, who knows the secret of the transformation, can read the original message.

The message to protect is usually called the plaintext and the protected message is called the ciphertext. The process of transforming a message from plaintext to ciphertext is called encryption. The opposite procedure, from ciphertext to plaintext, is called decryption. To encrypt or decrypt a message we use a piece of secret information, the key. The security of a message is related to the knowledge of the key, which means that the management of the key is one of the most important aspects for the security of the encrypted data.

There is a famous maxim in cryptography that claims: "A cryptosystem should be secure even if everything about the system, except the key, is public knowledge". This quote is known as Kerckhoffs’s principle [1] in honour of Auguste Kerckhoffs, a Dutch linguist and cryptographer who was professor of languages at the École des Hautes Études Commerciales in Paris in the late 19th Century. Kerckhoffs’s principle was reformulated by Claude Shannon as “The enemy knows the system”. In that form, it is called Shannon’s maxim – in contrast to “security through obscurity”, it is widely embraced by cryptographers.

Kerckhoffs’s principle has some interesting correlations with the open source philosophy, where the algorithm (the source code) is public (free to share). As Bruce Schneier writes, “[a]s a cryptography and computer security expert, I have never understood the cur-
rent fuss about the open source software movement. In the cryptography world, we consider open source necessary for good security; we have for decades.”

This is an important aspect of the security of a cryptographic system: never use closed source algorithms. If you want to implement a cryptographic system that is reasonably secure, you should always use standard algorithms. Don’t spend time and energy to create a new cipher. If you are not an expert in cryptography, i.e. at least a competent mathematician, you will fail for sure. One of the most used standards in cryptography today is the Advanced Encryption Standard (AES [2]). This algorithm won the NIST competition in 2001 after five years of challenging with fifteen different designs.

Symmetric-key and asymmetric encryption

In cryptography we have basically two families of encryption algorithms: symmetric-key and asymmetric (or public key). The symmetric algorithms use a single key to encrypt and to decrypt data. The asymmetric algorithms use two keys: one to encrypt (the public key) and one to decrypt (private key). Asymmetric algorithms are typically used in communications systems because they can be used by remote users without the need to share a secret (key). Using a symmetric key we have to share the secret key between users. This can be a problem if we have an insecure communication channel. For instance, how do we share a secret key over the internet?

Using an asymmetric algorithm, we bypass that problem. We just encrypt the message with the public key of the receiver and the receiver can decrypt the message using his private key: we don’t need to share any secrets. The public key can be sent in plaintext, and if someone intercepts the key he/she cannot use it to decrypt the message. Moreover, it is not possible (or, at least, very, very difficult) to obtain the private key from the public key. The public key and the private key are related by a mathematical property, the integer factorisation problem [3]. The integer factorisation or prime factorisation is the decomposition of a composite number into smaller non-trivial divisors, which when multiplied together equal the original integer. When the numbers are very large, no efficient, non-quantum integer factorisation algorithm is known; an effort concluded in 2009 by several researchers factored a 232-digit number (RSA-768) utilising hundreds of machines over a span of two years.

But, even if symmetric ciphers have the problem of sharing the key, they are the most widely used algorithms to encrypt data. The main reason is because they are very fast, compared with the asymmetric algorithms. If we have problems sharing the key we can use the asymmetric algorithms to encrypt a session key used by a symmetric algorithm to encrypt the data. This is the hybrid method, using asymmetric and symmetric algorithms, and it’s very common in the implementation of cryptographic systems (one of the most famous protocols that uses this technique is the OpenPGP standard, RFC 4880 [4]).

Hash functions

The algorithms shown so far are able to protect sensitive information by transforming the plaintext into a ciphertext. If we want to protect data against an accidental or intentional change, we can use a special family of algorithms: hash functions. Hash functions take an arbitrary block of data and return a fixed-size bit string, the hash value (sometimes called the message digest or simply digest). These functions are used in different computer science fields. For instance, they are used with the hash table data structure to calculate the key of a value. In cryptography, a hash function has four main or significant properties:

- it is easy to compute the hash value for any given message;
- it is infeasible to generate a message that has a given hash;
- it is infeasible to modify a message without changing the hash;
- it is infeasible to find two different messages with the same hash.

If we generate the hash of a string and if we change just a single bit of the string, the hash value will change completely. This property can be used to prevent change and manipulation of the original data. Hash functions are used in cryptography to authenticate messages, to generate digital signatures, to generate checksums, etc.

Cryptography vs. security

Cryptography doesn’t mean security – this is a very important aspect to consider in the design of a software system. Frequently, cryptography is wrongly associated with security. If we hear the word cryptography, we think that the system is secure by default. That is not true.

Cryptography is only an instrument that can help to improve the security of a system. At the same time, if we don’t know how to use this tool we can do a lot of damage to the security of a system, instead of improving it. For instance, if we implement a weak algorithm to encrypt information, or if we use a very simple password as key of a cipher, we are not improving the security of a system for sure. As with any tool, you have to know how to use it in the right way. Of course, it is not enough to read an article like this, or even an entire book about cryptography to be sure about the security of your design. Security is a dynamic property of a system, it changes day-by-day. It is very important to keep up-to-date: every day, researchers and hackers discover new security flaws, even using cryptography.
Of course, using standard algorithms we can reduce the risk of attacks, but we cannot remove it. If you want to design a secure system, you should never stop studying and keep updated with the last news about computer security. This is particularly true for web applications where the possible attacks can come from one or millions of people.

PHP has not been considered "enterprise ready" for many years, but now it is. Companies around the world use PHP as a language to build web applications that are used by millions of people, and security is a crucial aspect of these applications. Cryptography can help with secure design and PHP offers many way to use it.

**Cryptography in PHP**

PHP offers different implementations of the most important cryptographic algorithms. In particular, PHP has the following cryptographic extensions:

- Hash
- Mcrypt
- OpenSSL

The **Hash** extension requires no external libraries and is enabled by default as of PHP 5.1.2 – it replaces the old mhash extension. With this extension, we can generate hash values or HMACs (Hash-based Message Authentication Codes). Hash supports the most common hash algorithms used in strong cryptography. To find out which algorithms are supported by your PHP environment, you can use the function `hash_algos()`, which gives a list of all the algorithms supported.

The **Mcrypt** extension is an interface to the mcrypt library [5] which supports a wide variety of block algorithms such as DES, 3DES, Blowfish (default), 3-WAY, SAFER-SK64, SAFER-SK128, TWOFISH, TEA, RC2 and GOST in CBC, OFB, CFB and ECB cipher modes. This is the most popular extension in PHP for encrypting data using symmetric ciphers.

The **OpenSSL** extension uses the functions of the OpenSSL project [6] for generation and verification of signatures and for sealing (encrypting) and opening (decrypting) data. You can use OpenSSL to protect data using public key cryptography with the RSA algorithm.

**How to encrypt with AES-256**

We are going to show how to encrypt data using the AES algorithm with a 256-bit key (AES-256). The AES algorithm is implemented in the Mcrypt extension.
AES uses a block size of 128 bits to encrypt data, the plaintext is split into blocks of 128 bits and each block is encrypted with the AES algorithm. Mcrypt offers six different modes to encrypt and concatenate each block – they are: CBC, OFB, CFB, ECB, NOFB, and STREAM.

In this article we will show only Cipher-Block Chaining (CBC), which is one of the most used, as suggested by the authors of the famous book Cryptography Engineering [7]. There are other modes of operation for cryptographic block ciphers, like CCM and OCB, but these are not supported by Mcrypt and they are too advanced for the scope of this article.

The Cipher-Block Chaining mechanism is described in Figure 1.

The first block of the plaintext (Block 1, in white) is combined with the Initialisation Vector (IV) using the XOR operator \( \oplus \) in the diagram. The IV is a salt value used to improve the randomness of the encryption schema. It’s not a secret and can be sent in plaintext – in fact, it’s usually sent together with the ciphertext. The result of the XOR operator is encrypted using a symmetric cipher (in our case the AES algorithm), with the key. The encrypted block (Block 1, in grey) is the first block of the ciphertext. CBC mode reuses the encrypted Block 1 for the second block. The encrypted Block 1 is combined with the Block 2 of the plaintext using the XOR operator. The result is encrypted using AES and the key. This mechanism is repeated for all the blocks of the plaintext.

In the Mcrypt extension of PHP, AES is reported with the original name of the algorithm, Rijndael. There are three different implementation of Rijndael in Mcrypt:

- MCRYPT_RIJNDAEL_128
- MCRYPT_RIJNDAEL_192
- MCRYPT_RIJNDAEL_256

They refer to the different sizes of the block (128, 192, and 256). Because the AES standard uses a block size of 128 bits, if we want to be compliant with the standard we have to use Mcrypt’s MCRYPT_RIJNDAEL_128 algorithm.

To encrypt a string value we can use the mcrypt_encrypt() function. Here’s an example with AES-256 in CBC mode:

```php
$ciphertext = mcrypt_encrypt(
    MCRYPT_RIJNDAEL_128,
    $key,
    $plaintext,
    MCRYPT_MODE_CBC,
    $iv
);
```

Where $key is the key of the encryption algorithm (which must be 256 bits), $plaintext is the plaintext to encrypt and $iv is the Initialization Vector. To decrypt, we can use the mcrypt_decrypt() function, which uses the same parameters as the encrypt function, where $plaintext becomes $ciphertext.

The usage of mcrypt_encrypt() and mcrypt_decrypt() is straightforward, but is it enough to guarantee the security of the information that we want to protect? Unfortunately, no – we have to add another process, integrity authentication.

We need to add an integrity authentication schema to discover possible manipulations of the ciphertext. Without that schema an attacker can easily manipulate the ciphertext and try to attack the implementation using cryptanalysis techniques. In fact, in 2002, Serge Vaudenay, a French cryptographer, discovered a way to decrypt a message encrypted in CBC mode without needing the key, an attack known as the Padding Oracle Attack [8]. This attack has been used against web applications built with Java, Ruby on Rails and ASP.NET.

To protect an encrypted message against the Padding Oracle Attack, we can use one of the following techniques:

- authenticate-then-encrypt;
- encrypt-then-authenticate;
- encrypt-and-authenticate.

We will use the encrypt-then-authenticate schema because, combined with CBC mode, it is considered secure (see references [7] and [9]).

How can we authenticate the encrypted messages? We can use an HMAC. HMACs are hash functions that work with a secret key. In order to generate the same hash value for a specific string, we have to use the same key. PHP offers the hash_hmac() function to generate HMAC values. The syntax of hash_hmac() is as follows:

```php
string hash_hmac (string $algo, string $data, string $key [, bool $raw=false])
```
Cryptography in PHP

Full example in PHP of how to use AES-256 with encrypt-then-authenticate schema. Listing 1 shows an example using a PHP class.

The encrypt() method of the AES class has the following parameters: $data is the plaintext, $key is the secret key and $iv is the initialisation vector (optional). If we don’t pass the $iv parameter encrypt() will generate a random value using the mcrypt_create_iv() function of Mcrypt. This function generates a random IV based on the MCRYPT_DEV_URANDOM seed (read data from /dev/urandom on GNU/Linux systems).

To be sure to use a key of 256 bits, we hash the value of $key using the SHA256 algorithm. We store this hashed key in the $hashKey variable, using the binary output to have the full ASCII set of characters for the password. $hashKey is the key used in the AES cipher. This is a good practice in cryptography – never use a user’s password as the key of a cipher.

Listing 1

class AES {
    /**
     * Encrypt
     *
     * @param string $data
     * @param string $key
     * @param string $iv = null
     * @return string
     */
    public static function encrypt($data, $key, $iv = null) {
        if (empty($iv)) {
            $iv = mcrypt_create_iv(16, MCRYPT_DEV_URANDOM);
        }
        // generate a 256 bit key from $key
        $hashKey = hash('sha256', $key, true);
        // PKCS #7 padding
        $pad = 16 - ((strlen($data) % 16);
        $data = str_repeat(chr($pad), $pad);
        $encrypted = mcrypt_encrypt(MCRYPT_RIJNDAEL_128, $hashKey, $data, MCRYPT_MODE_CBC, $iv);
        // HMAC using md5($key) as key
        $hmac = hash_hmac('sha256', MCRYPT_RIJNDAEL_128 . $iv . $encrypted, md5($key));
        return base64_encode($iv) . $hmac . base64_encode($encrypted);
    }
    /**
     * Decrypt
     *
     * @param string $data
     * @param string $key
     * @return string
     */
    public static function decrypt($data, $key) {
        $iv = base64_decode(substr($data, 0, 24));
        $hmac = substr($data, 24, 64);
        $encrypted = base64_decode(substr($data, 88));
        $hmacNew = hash_hmac('sha256', MCRYPT_RIJNDAEL_128 . $iv . $encrypted, md5($key));
        if (!$self::compareString($hmacNew, $hmac)) {
            return false;
        }
        $hashKey = hash('sha256', $key, true);
        $plaintext = mcrypt_decrypt(MCRYPT_RIJNDAEL_128, $hashKey, $encrypted, MCRYPT_MODE_CBC, $iv);
        // remove PKCS #7 padding
        $pad = ord($plaintext[($len = strlen($plaintext)) - 1]);
        return substr($plaintext, 0, strlen($plaintext) - $pad);
    }
    /**
     * Compare two strings to avoid timing attacks
     *
     * @param string $stringA
     * @param string $stringB
     * @return boolean
     */
    public static function compareString($stringA, $stringB) {
        $stringA = (string) $stringA;
        $stringB = (string) $stringB;
        if (strlen($stringA) !== strlen($stringB)) {
            return false;
        }
        $result = 0;
        for ($i = 0; $i < strlen($stringA); $i++) {
            $result ^= ord($stringA{$i}) ^ ord($stringB{$i});
        }
        return $result === 0;
    }
}

www.webandphp.com
AES is a block cipher: the plaintext is split into blocks of 16 bytes. The plaintext has to be padded to bring them to this length. One method is to fill out the last block with a 1-bit followed by zero bits (zero padding). The Mcrypt extension uses the zero padding mechanism by default. In our example we use PKCS7 padding, where the value of each added byte is the number of bytes that are added [10]. PKCS7 padding is particularly useful during the decryption phase because we can easily remove the padding value from the message.

After the encryption, we generate the HMAC value using the input string composed by the algorithm name, the IV and the encrypted message. The secret key used in the HMAC function is md5($key). Theoretically, it’s more secure to use a complete different key for the HMAC. We used a different key, generated with the MD5 algorithm, based on $key.

The result of HMAC, the IV and the encrypted message is the result of the encrypt method. As you can see, we encoded the output in Base64 format to avoid encoding problems on different platforms. The Base64 format is a universal format composed of 64 characters (a-z, A-Z, 0-9, +, /) that works fine in all the operating system environments.

The decrypt() method of the AES class has only two parameters: $data, the ciphertext, and $key, the secret key. The IV is not needed because it’s stored at the beginning of the ciphertext. We can easily retrieve the IV, the HMAC and the encrypted string from the ciphertext because we know the size of each component (24 bytes for the IV and 64 bytes for the HMAC).

Before we decrypt the ciphertext we have to check that the encrypted message has not been compromised. We calculate the HMAC based on the algorithm used, the IV and the encrypted message and we compare the result with the HMAC stored in the ciphertext. If the result is the same, we can proceed with the decryption of the message; otherwise, we return false.

To compare the two strings $hmacNew and $hmac, we use a special function instead of using the === operator. This is because we want to prevent timing attacks. A timing attack is a side-channel attack in which the attacker attempts to compromise a cryptosystem by analysing the time taken to execute cryptographic algorithms. Every logical operation in a computer takes time to execute, and the time can differ based on the input; with precise measurements of the time for each operation, an attacker can work backwards to the input.

This attack may sound like something out of a Mission Impossible movie, but it is very common in the real world, so we have to take care of it. To have an idea of how the timing attack works in the real world, you can read the article referenced in [11]. The compareString() function reported in the AES class is able to prevent timing attacks because it always uses the same computational time to compare two strings.

How to store a password using bcrypt
One of the most-used techniques to store a password is to use a hash function with a salt value (where the salt is a random string). Well, this method is not secure any more. Why? Because with modern CPUs and cloud computing systems, it is relatively easy run a brute-force attack, even if you use a salt value.

In 2010, Thomas Roth, a security researcher, showed how to crack an SHA-1 password of 6 characters in 49 minutes using a cluster of Amazon EC2 instances, for a total cost of $2.10. The attack is detailed in reference [12].

The problem with hash functions, like the SHA family, is that they are too fast. If we want to prevent brute-force (or dictionary) attacks we have to use algorithms that are computationally slow. For instance, we can use algorithms like PBKDF2 or bcrypt. These algorithms are basically adaptive hash functions that are very, very slow.

The bcrypt algorithm is included by default in PHP 5.3. In order to use it, we have to call the crypt() function with a particular parameter. Here’s a snippet of code showing how to use it:

```
$salt = substr(str_replace(‘+’, ‘.’, base64_encode($salt)), 0, 22);
$hash = crypt($data, '$2a$' . $workload . '$' . $salt);
```

Where $salt is a random string, $data is the information that we want to protect (for instance, a user’s password) and $workload is a parameter of the bcrypt algorithm which we will come back to later.

Bcrypt is was designed by Niels Provos and David Mazières and was presented at USENIX in 1999 [13]. The algorithm is an adaptive hash function based on the Blowfish symmetric cipher. Besides incorporating a salt to protect against rainbow table attacks, bcrypt is an adaptive hash: over time, it can be made slower and slower so it remains resistant to specific brute-force search attacks against the hash and the salt.

Earlier we mentioned a $workload parameter. This is the cost of the algorithm, an integer value that specifies the workload. A greater value in $workload means more work for the algorithm, which means more computational time and consequently more security. The $workload parameter of the PHP crypt() function is an integer value from 10 to 31.

The question is: which value to use for the $workload? The answer depends on the speed of the CPU that you are going to use. For instance, running the bcrypt PHP implementation on my laptop, an Intel Core 2 at 2.1Ghz, the computational speed is as follows:
A typical output from a bcrypt hash is something like this:

```
$2a$14$c2Rmc2Fka2hmamhzYWRmauBpwLDFKnpTmCeu4HvINvMaLrNIF20
```

Where the text in blue is the bcrypt specification with the workload (14), the text in red is the salt value and the text in black is the hash value. The salt value can be stored in plaintext because it is not a secret. The salt is a parameter that improves the randomness (and therefore the security) of the bcrypt algorithm.

### Public key cryptography in PHP

So far we have showed how to use symmetric ciphers and hash functions in PHP. Now we want to introduce how to use public key cryptography. We will show how to generate a private key and a public key using the OpenSSL extension of PHP [14] and how to encrypt and apply a digital signature to a string.

#### Generate private and public keys

We can generate a new private and public key pair using the following code:

```php
$privateKey = openssl_pkey_new(array(
    'private_key_bits' => 1024,
    'private_key_type' => OPENSSL_KEYTYPE_RSA,
));
openssl_pkey_export_to_file($privateKey, '/path/to/privatekey', $passphrase);
// get the public key
$details = openssl_pkey_get_details($privateKey);
file_put_contents('/path/to/publickey', $details['key']);
```

In this example we generated a RSA private key of 1024 bits ($privateKey) and we stored it in a file using a password ($passphrase). This password protects the private key against illegal access of the file. Without the $passphrase, it is not possible to read the content of the private key. Moreover, we retrieved the public key ($details['key']) and we stored it in a file. The public key is not protected with a password because the access must be free.

#### Encrypt with the public key

Now we can start to encrypt/decrypt data using the RSA algorithm. For instance, suppose we want to send an encrypted message to Bob and we have the Bob’s public key, we can use the following code:

```php
$pubKeyBob = openssl_pkey_get_public("file://public.key");
openssl_public_encrypt($message, $ciphertext, $pubKeyBob);
```

The syntax ‘file://’ is mandatory to read the key from a file. The encrypted data is stored in the $ciphertext variable. The ciphertext can only be decrypted using Bob’s private key. That means that only Bob can decrypt the message.

#### Decrypt with the private key

Bob can decrypt the message using the following code:

```php
$privateKeyBob = openssl_pkey_get_private("file://private.key", $passphrase);
openssl_private_decrypt($ciphertext, $message, $privateKeyBob);
```

$cipherertext contains the message to decrypt and the decrypted message is stored in $message. The $passphrase is Bob’s password for the private key.
Apply a digital signature

A public key cipher can also be used to authenticate a message. If we want to be sure that a message has been generated by a specific user, we can use the sender’s private key to apply a digital signature to it. This is done with the usual encryption method, where we use the private key instead of the public key. In this way, everyone can decrypt the message, using the public key of the sender, in order to verify that the message comes from the legitimate owner. Moreover, this digital signature also guarantees the integrity of the message. We can be sure that the message comes from the sender and that it has not been changed by anyone.

Because public key algorithms are very slow, it’s a common practice to generate the hash of the message and use the hash digest instead of the message. In this way, we can have the same result using less CPU time. In fact, if we change just one bit in the original message, the hash value will be different and the digital signature will fail.

PHP’s OpenSSL extension offers a set of APIs for digital signing. The openssl_sign() function computes a signature for the specified data by using SHA1 for hashing followed by encryption using a private key. To verify a digital signature we can use the openssl_verify() function. The following code shows an example of how to generate the digital signature of a file and how to verify the signature:

```php
$message = file_get_contents('path/to/message');
$privateKey = openssl_pkey_get_private('file://private.key', $passphrase);
openssl_sign($message, $signature, $privateKey);

$publicKey = openssl_pkey_get_public('file://public.key');
echo (openssl_verify($message, $signature, $publicKey)===1 ? "OK" : "ERROR");
```

The digital signature of the message is stored in $signature. To verify the digital signature, the openssl_verify() function returns 1 if the signature is correct, 0 if it is incorrect and -1 on error.

It is important to note that, in this example, the message is not encrypted. If you want to implement an encryption schema plus the digital signature, you have to apply the digital signature first and encrypt the message afterwards. This means using the sender’s private key to generate the signature of the message and the receiver’s public key to encrypt the message. In the end, the message will be composed by concatenating the ciphertext and the signature.

Conclusion

In this article, we introduced the use of cryptography in PHP. We showed how to encrypt and decrypt sensitive data using symmetric and asymmetric algorithms. Moreover, we discussed hash functions and how to use the bcrypt algorithm to safely store users’ passwords. We showed that cryptography doesn’t necessarily mean security and that it is always better to use standard algorithms like AES, SHA or RSA. If we build software that is reasonably secure today, in the next few months or years it probably won’t be. Computer security is always evolving, especially in the field of web applications, and you should always stay on the alert.

References


Enrico Zimuel (enrico@zimuel.it) has been a software engineer since 1996. He works as a Senior PHP Engineer at Zend Technologies, in the Zend Framework Core Team. He is an international speaker and author of books and articles about computer security, cryptography and web programming. He has done research in computer science at the Informatic Institute of the University of Amsterdam and at the Science Department of the G. D’Annunzio University in Pescara, Italy.