WHY YOU CAN BUY YOUR APPLICATION SECURITY AND THEN BUILD WITH IT.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>3</td>
</tr>
<tr>
<td>Buy or build? Why not both!</td>
<td>5</td>
</tr>
<tr>
<td>Development</td>
<td>6</td>
</tr>
<tr>
<td>Buying a legacy solution</td>
<td>8</td>
</tr>
<tr>
<td>Maintenance</td>
<td>8</td>
</tr>
<tr>
<td>Operations</td>
<td>10</td>
</tr>
<tr>
<td>2FA availability for the application</td>
<td>10</td>
</tr>
<tr>
<td>2FA usability for the user</td>
<td>11</td>
</tr>
<tr>
<td>Security</td>
<td>12</td>
</tr>
<tr>
<td>Secure your solution with Authy</td>
<td>14</td>
</tr>
<tr>
<td>Summary</td>
<td>17</td>
</tr>
</tbody>
</table>
INTRODUCTION

When building any software application, making it secure is a top priority. But security software is very complex and very risky to build yourself. Do you task your development team to design their own or do you buy it from a vendor?

Consider you are a company that builds commercial aircraft. Your team of designers and engineers are experts in aerodynamics, passenger capacity and aircraft assembly. But, when it comes to the engines that keep your aircraft aloft, do you ask those same engineers to design and build the complex turbojets? Or should you buy from a company with highly skilled jet engine engineers that has a history of manufacturing and maintaining reliable power plants?

The exact same thing can be said for making sure the software applications you build are secure. Asking your team of developers to build the security that protects and powers your solution is risky, not to mention an unnecessary time and resource drain, taking your team away from their core value.

You may be experts in the financial, social, gaming or other industry specific features of your software. But are you well versed in, or up to date on, implementing important security mechanisms? Security requires expertise of its own. It is a critical piece in the construction of applications and unless you already have a team of highly trained security developers, you don’t want to be taking the risks of building your own solution, keeping it current and monitored. You need your application to fly safely and the doors to be securely locked, and the best way to do that is to buy mature and trusted security solutions.

But this buy versus build subject isn’t always clear cut. Many people think that buying a solution means building a server, installing software and configuring it, which results in costly long term maintenance of the operating system, database and vendor’s software. Legacy solutions are also black boxes with some dials and buttons to configure them. Meaning you have little to no control over the logic flow and user experience. However this was true in the 1990’s, but not with today’s massively scalable and API driven cloud services. Investing in software exposed via a cloud API allows you to buy and THEN build. You have full control over how the technology integrates into your application, without the headache of running and maintaining the infrastructure.

When looking at adding improved security to your application, you quickly learn it is a very broad subject. Data encryption, SQL injection, transport encryption and denial of service (DoS) attack protection are just a few of the areas you need to address. One of the most common
and immediate problems any application developer faces is protecting the front door to your application, the login. Everyone knows using just a username and password is not enough to secure application access and yet this is still the most commonly used method. The problem is not really the idea of passwords, but rather that people choose to use passwords that are appallingly vulnerable or use the same password across many accounts. Passwords can however still be effective when paired with other information and protocols. In the security industry we refer to these extra pieces of information as factors, and they break down into three main categories:

- **Something you know**, i.e. something knowledge-based; this includes familiar and traditional security methods like your password, username, PIN, and security question responses like your mother’s maiden name or the last 4 digits of your social security number.

- **Something you have**, i.e. a device, smart card, or USB token. Many companies have used small credit card-sized devices (smart cards) that store some information which is used as part of a login. Laptops and desktops would then have a smart card reader that means you can only login if you have the smart card in your hand.

- **Something you are**, a trait of your human self. This is usually in the form of biometrics and ranges from scanning your iris to the recently popularized Apple Touch ID that examines your fingerprint.

Using more than one piece of information to secure access to applications is called two factor (2FA) or multi factor authentication (MFA). 2FA technologies have been around for a while, in the 80’s and 90’s they typically involved an expensive 2FA device that was shipped to every employee. Nowadays the common smartphone can perform the same 2FA role.

Many 2FA projects begin under the belief they can be integrated for free and/or are trivial to build. In practice, neither is true. Making the age old question of whether to “Buy or Build” more critical than ever.

This paper provides evidence that all 2FA solutions are not created equal, and that the commonly used Google Authenticator, and other software based solutions, vary significantly in their cost and capabilities. Many technologies are expensive to run, from the impact to your budget to the quality of the user experience. Therefore, it is important to fully research your options and understand the market of 2FA products before you embed such a critical and visible feature into your business.
BUY OR BUILD? WHY NOT BOTH!

In the late 1990’s people who built computer software programs were called programmers. The expectation was that these programmers wrote nearly all of the software code for their applications. In the games industry, programmers had to write custom code for each vendor of graphics card and business software programmers wrote code specific to the databases they were using. But today, the role of programmer has changed dramatically. Instead of building the entire application from scratch, new software is built on platforms and leverages existing frameworks. Layers and layers of software, with built-in functionality to handle networks, databases, user interface and more. We no longer have to keep reinventing the wheel. So much progress has taken place in software development, that you can now put together a simple web-based application, with a database, application server and a whole framework for writing the user interface, within a matter of hours.

Programmers used to just write code, but developers, who do still write code, build applications. The modern software developer chooses a platform/framework on which to create their solution. They spend most of their time writing only the code that is specific to their business. These platforms are built and maintained by a new group in the software world, called DevOps. They are developers who build the systems on which the application software operates. But the development part of DevOps is about integrating other services to the platform to increase reliability, scale and implement core functionality to the platform provided by other vendors.

So when DevOps needs to monitor the state of the application? They buy and integrate a service. Need to ensure your service can scale when you acquire more business? Buy and integrate a solution. Need to ensure your application is using the most user friendly and secure two factor authentication? Like other aspects of your business software platform, it makes sense to buy and integrate a solution. Buying these core services means your developers spend their time actually working on the functionality that makes your service valuable. Buying might seem like a costly route, but as we discuss throughout this document, building such core components yourself is a lot more expensive and risky.

But as we mentioned at the start of this document, don’t confuse buying your software with the types of solutions available on the market 10-20 years ago. Not only has the role of a programmer changed, but also the nature of how software is delivered to you. Cloud based software that is designed to be integrated into your own applications is delivered via (usually RESTful) APIs. These APIs are typically much simpler than the older COM and SOAP style APIs and therefore are quickly integrated into your solution. This results in the best of both worlds. You buy the API and then build it into your applications.

```java
public static long getCode(byte[] secret, long timeIndex)
        throws NoSuchAlgorithmException, InvalidKeyException {
    SecretKeySpec signKey = new SecretKeySpec(secret, "HmacSHA1");
    ByteBuffer buffer = ByteBuffer.allocate(8);
    buffer.putLong(timeIndex);
    byte[] timeBytes = buffer.array();
    Mac mac = Mac.getInstance("HmacSHA1");
    mac.init(signKey);
    byte[] hash = mac.doFinal(timeBytes);
    int offset = hash[19] & 0xf;
    long truncatedHash = hash[offset] & 0x7f;
    for (int i = 1; i < 4; i++) {
        truncatedHash <<= 8;
        truncatedHash |= hash[offset + i] & 0xff;
    }
    return truncatedHash %= 1000000;
}
```
DEVELOPMENT

First let’s look at the effort involved in incorporating 2FA into your application by building it all yourself. One of the most common factors of 2FA is a token or one-time password (OTP). Generated during login and sent to you via SMS or it’s generated offline using the current time and a shared cryptographic secret (Time based OTP, or TOTP).

After the user provides their username and password as the first factor, they next enter the token and you have to decide about how to deliver the token to them out-of-band. This means the application they are logging into can’t just display the token, you need to get that information to them via another channel and ideally on a separate network or device from the one being used for the login. Usually their phone.

Do you use SMS as a way to send that token at the time of login? Or do you create your own smartphone app to generate it? To create the token, do you find an open source library that implements the TOTP standard? Which one, how often is it maintained and how secure is it?

Let’s go into a little detail on some of these questions. If your user base is truly global, not everyone is going to have a smartphone that can run an application. Therefore, you need to ensure you can get the one-time password to them without an app. Nearly every mobile phone supports SMS, so that is a great way to get the token to the end user. But now you need to find an SMS provider. Which one? Does the SMS provider you choose have good global coverage? Do you need to choose two providers and build in logic to fail over if one is unable to deliver the code to the region your user is in? What happens if the user loses their phone and their number changes, how do you handle that change? What if the user hasn’t got access to an SMS enabled device? Then what? You could just call the user and use a voice library to speak the token to them. But how? Do you build your own text to voice app? How do you then create the phone call connection from your software to the user’s phone? What about translating the voice into the language used by the end user?

The user might have a smartphone which can install apps from an online store. Do you write your own 2FA software app? Or do you use one that someone else has created that implements the standard OTP protocols? Google was one of the first companies to offer a 2FA service to their millions of users and they developed the Google Authenticator smartphone app. It supports the TOTP standard, so you could just tell users to install that. But what if you want to brand the user experience? How do you add your company logo so end users are assured they are using the right token for your website?
You might already have a mobile app and want to embed the 2FA features into it. There are quite a few libraries for TOTP you can install on your web application, but finding code libraries for iOS and Android is a lot harder. You can, of course, just write the code yourself. Generating the code in a mobile app is also only one aspect of what you need to do. What happens if a user loses their phone? The cryptographic secret that generates the codes is now lost for the user, how do they recover? What if the user wants to use more than one device? Do you require they go through the same process each time they want to use a device for 2FA?

2FA is always improving security features as well as improving the end user experience. Recently, companies like Yahoo and CapitalOne have implemented a push notification based solution for security. Instead of a user re-typing in the token, their phone gets a notification and the user responds and approves or denies the activity that just took place, for example a login. Implementing such a feature requires understanding how the Apple Push Notification Service (APNS) and the Google Android Cloud Messaging (GCM) services work. Plus all the free TOTP authenticator clients don’t support these new push notifications. So now what? Do you tell your users to use two apps for your 2FA? Google Authenticator for when they need to generate a token offline and another one you build that supports push notifications? Most likely you’d develop your own 2FA client with both push and TOTP capabilities. But now your old users need to migrate to it.

Not only do all the above options result in a lot of code and dependencies. But implementing them often exposes you to the inner workings of security protocols. Are you or your developers experts in security software development? Remember, if you make a mistake in this part of your application, you are impacting the front door to your business. If the 2FA solution you implement breaks, nobody can login. If it has a vulnerability, it’s no longer a security feature but now an avenue for a security breach.

It doesn’t make sense today for your skilled developers to be spending time researching, understanding and implementing lots of code for 2FA. Specialist security software developers are hard to find and can be expensive. You want your developers to be adding value to your application logic and not spending time implementing security features which should be just part of the platform.

In summary, developing your own 2FA solution is not simple. It requires specific knowledge and can take significant time to implement.
BUYING A LEGACY SOLUTION

We mentioned earlier that you could buy your 2FA, but instead of a modern cloud API based solution there are older legacy technologies you can deploy. These solutions were designed and architected in the days when all software was sold on a CD and installed onto servers. It was mainly designed for the enterprise IT groups, who typically don’t have developers on staff, and therefore, instead of exposing flexible APIs, they delivered configurable interfaces that you sat in front of your application.

This may seem like an attractive option, but you lose control over the login flow and be limited in the login interface. This results in a poor user experience and can make accessing your service more painful. You still end up with a lot of infrastructure to maintain, update and monitor. This can be just as costly in the long run as building it yourself.

So the ideal solution would be a balance. Something you can integrate into your application logic and therefore retain full control over the whole user experience. Yet you need to deliver this without months of work and hundreds of lines of code that need to be constantly maintained. A solution that continues to evolve as 2FA technology improves, yet doesn’t require you spend more weeks of development to upgrade.

MAINTENANCE

Let’s say that you took the plunge and used an open source 2FA library to generate codes and are using an SMS provider to deliver them to users at login. You’ve added features to your website to allow users to configure 2FA and you’ve recommended Google Authenticator. What next?

You need to make sure your implementation stays up to date. The code and service needs to be maintained. At writing, there are 290 repositories on GitHub offering pre-built 2FA functionality. 23 of them in the last 30 days have had updates. One of the most popular Python libraries has had 10 updates in the last 3 days. So if you choose to use one of these libraries, you will need to update your solution on a fairly regular basis. But confusingly, the opposite is also true. Of the top 10 most popular repositories, two have not had an update since 2013! So you could end up choosing a library which isn’t updated and therefore you might be stuck with a codebase you rely on that’s lacking in new features or contains security vulnerabilities due to lack of focus from the open source community.
If you are relying on an open source library for your 2FA, you either need to be on top of updates and ensuring your own codebase is using the latest version or, at the other end of the scale, you run the risk of using a library that has little activity and has not been updated. If you decided to implement both SMS and voice as a way to deliver the 2FA tokens, you may well have more libraries from different vendors that are part of your 2FA solution. You may end up with multiple libraries, all with different development cycles, that need to be kept up to date.

These issues relate to the server side of your application. If you decided to implement your own client application, you have at least two smartphone apps to maintain. One for Apple iOS and another for Google Android. Depending on demands of your customers, you may also need to build a Windows phone/desktop application. Major releases of these operating systems take place on an annual basis with several intermittent patches and often require you to update your application to address fixes and reflect user interface changes or implement new features like Apple’s Touch ID or integrate with additional devices like the Apple Watch.

Or you may have just relied on existing 2FA client applications like Google Authenticator. Now you are at the mercy of when Google decides to update their app. Worse still, you have little to no control over bugs in other people’s software, especially when you have no commercial agreement with them. Google once updated their app and it wiped the tokens from everyone’s devices. (http://techcrunch.com/2013/09/04/dont-install-the-google-authenticator-for-ios-update-unless-you-want-your-stored-user-accounts-wiped/)

These are all problems of maintenance based on the market today. But 2FA is becoming more and more popular, this means new techniques and enhancements are going to be created. Therefore, if you have built your own implementation 2FA, the maintenance of the code is only going to increase over time. This works against the mentality of most modern application development environments. DevOps want your core developers to choose other vendors to deliver and maintain critical parts of the infrastructure which runs the business. You need to choose a 2FA solution that minimizes the impact to your long term maintenance, but yet still remains current in features and security.
OPERATIONS

Implementing 2FA and maintaining the code are what your developers are doing. But DevOps has to keep the entire service running. 2FA is at the start of your user’s experience when they login, you need to ensure the service stays operational. If 2FA fails to work, your users cannot access your service. That’s critical. Some would argue that your business being offline is actually worse than it being insecure.

Operationally there are two main areas to ensuring a smooth and quick 2FA experience. First there is the availability and performance of the 2FA technology to your application itself. Secondly, users are able to manage their own account and their 2FA devices.

2FA availability for the application

Going into detail on the application side, let’s look at just SMS. An important feature to have in any 2FA solution, enabling security for those without the ability to download and install a smartphone app. Every time a user is prompted to login, they need to get the token via SMS and if you’ve built the 2FA solution yourself, you need to use an SMS provider. There are on-premises SMS solutions, but most modern applications will want to use cloud based SMS services. Yet not all SMS providers are the same. First, each SMS provider will have different delivery challenges. Some may have great coverage in North America, but have poor performance in Europe. You may find that if you always send the SMS from the same number, you get shut down due to automated spam systems on the network. In that case, you need to then manage a pool of numbers from which to send the SMS. When a specific SMS carrier has an outage, what do you do? To ensure your application is not impacted, it makes sense to use more than one SMS vendor so you can switch if one is not available. If you are delivering 2FA to a global community of users, you may also need multiple SMS vendors so you can use the right one depending on the user. So your North America users get their 2FA token via a different SMS vendor than your European users.

If you have decided you want an improved 2FA user experience, you may decide to build your own push notification type solution. This means instead of asking the user to retype the 2FA token, they interact with a smartphone application. There are two phases to this type of 2FA, reaching out to the user’s device to notify them of the 2FA event and then providing a way for the smartphone app to respond to your application when the user accepts or denies the request. Reaching out to the user is done via a push notification service. For Apple devices you use the Apple Push Notification Service (APNS), or you can choose...
the Google Cloud Messaging (GCM) service which supports their Android platform, Chrome browsers as well as iOS. However APNS supports OSX whereas GCM does not. So do you implement both?

If you implement Google Cloud Messaging, it provides the ability for the client to also respond to your service. With the Apple solution, you need to build into your 2FA client the ability for it to talk back to your application after the user has selected accept or deny. All of this functionality requires services built into your application to both send and receive the notifications and their responses. With both Google and Apple, you need to manage certificates for notifications to iOS and OSX devices. If the certificate expires, your 2FA stops working. These methods of cloud to device communication are also relatively new, Google has already deprecated a previous Android Cloud to Device Messaging (C2DM) service which was shut down entirely at the end of 2015.

So even just keeping the 2FA service running is costly work. We didn’t even mention that you have to implement monitoring of your SMS to push notification provider and to build a really high quality solution, you may have to implement another vendor for failover. That option to failover also needs to be developed. More code! More work! It all means a lot of work for DevOps.

2FA usability for the user

The other side of running a 2FA solution is the end user experience. With SMS you need to know the user’s phone number to send the token to. Once you’ve captured that information, what happens if a user loses that phone? Do they have to wait until their phone provider replaces the phone before they can access your service? Do you provide a way for the user to recover access while they don’t have access to the phone? How do you do that securely? What if they change their phone number? How do they access your service to login and change the number when you are sending a text to the old number? In many cases a user won’t have access to devices with both numbers at the same time. What do they do? Getting this process right is critical, the profile of a hacker and a legitimate user very similar.

If you’ve implemented offline 2FA using a TOTP solution, it means a shared secret has been created and stored on their smartphone. This is usually done by secure process to bootstrap the key such as via a QR code captured by the device’s camera. This secret is then used to generate the tokens required to login. What happens if they lose their device? They can’t login to your application to generate a new QR code for a replacement phone and users are stuck. Do you build another system for allowing users to bypass the 2FA when they lose
their smartphone? Can you figure out a way to share the secret across multiple devices? Do you provide a support phone number for them to call and walk them through another set of security questions? Who provides this support? Do you already have a support team? Who trains them on making sure you implement the 2FA bypass process securely? Will it degrade security and become a backdoor?

Do you rely on the end user to choose their own 2FA client? There are over 25 different apps in the Apple store. Do you recommend all or only support one? Which? The most common is Google Authenticator. But what happens when Google Authenticator doesn’t work for your solution, do you ask Google to make changes specific to you?

While it might seem easy to add 2FA to your application, operating it long term is not trivial. There are a lot more aspects to a successful and easy to use 2FA solution than just implementing the OTP standard in your login. Delivery of tokens, managing multiple vendors (SMS, voice, push notification network) and handling end user support can amount to significant time and cost.

SECURITY

The final piece in the puzzle of building your own 2FA solution is securing the whole implementation. First, securely creating the 2FA authentication software, then secondly ensuring it’s correctly implemented. Implementing process around credential generation, issuance, expiry, revocation, emergency access, retries and lockouts etc – the whole 2FA lifecycle management. While 2FA is about improving the security to your application, your application is at risk if the 2FA service itself has a vulnerability. Making sure your 2FA code, the service, support processes and end user practices are secure is very important.

If you’re using an open source library, you need to be sure it’s active and up to date. Open source software can be very secure and well maintained, yet the 2FA libraries available on GitHub are varied. The security of an open source library may get some scrutiny by the volunteering public, but there is no rigor or consistency to it. The most highly rated TOTP project, written in Python, only has 3 contributors of which only one is active. Can you be assured this contributor is aware of all the possible issues with their code? What happens if this one person stops contributing are you on your own? Of course, implementing this library isn’t just about importing it into your project. You also need to write code to integrate it into your application logic. Introducing more moving parts into the security of your application.
Do you ensure the 2FA logic in your application is well secured? Do your developers have the knowledge to keep it up to date based on newly found methods of attacking 2FA or security vulnerabilities in general? In recent times many developers had to understand and react to a bug in the OpenSSL library called Heartbleed. Will your team be skilled enough to patch such issues in a timely fashion?

If you decide to build your own from scratch, you will end up heavily involved in cryptography. From the generation of the OTP tokens, to validating them, to securing storage of the keys and implementing public/private key mechanisms to secure server to client push notifications. This can be a hornet’s nest of complexity and trying to write or modify cryptographic functions, without expert knowledge, is often a really bad idea.

Which brings us onto the next problem with building your own 2FA service. For many hackers, targeting the 2FA service itself is a common way to gain illegitimate access to the application. Poorly designed or configured 2FA solutions can easily be circumvented by attacking the 2FA reset process. There are also methods of capturing the 2FA token by phishing. The attacker sends your users an email to a fake login page which catches the end users 2FA token and then in turn replays this against your own application. If you want to truly protect against this, you need to implement the latest push notification-type 2FA. Yet this is significantly more work and you still need to consider implementing a token based solution so that users without smartphones or cell phone connectivity can access your application.
SECURE YOUR SOLUTION WITH AUTHY

It’s clear the investment and risk in building your own 2FA is significant. It is for this reason Authy was created. It is a cloud-based API and service that removes the complexity and effort in adding 2FA to your application. Yet because it is an API, it gives you total control over how and when to implement 2FA into your application.

Yes, that does mean you are still writing code, but the difference is that the amount of code is significantly reduced, in some cases from hundreds of lines of code to tens. Authy hosts all the complexity in a cloud service with constant security testing and monitoring. We handle the SMS delivery via multiple providers, we also have voice and push notification. We have a whole team dedicated to the production, maintenance and security of the 2FA service, our clients, SDKs and code libraries. Our API is very streamlined. Most of our customers find their developers can integrate our API into their application in a matter of weeks.

Authy provides their 2FA service to millions of users and thousands of businesses. Companies like Dell, VMWare, Coinbase, FitBit, CloudFlare, Twitch.tv and SendGrid have all chosen Authy to strengthen the security of their platforms and applications. All this activity on our cloud platform enables us to learn a great deal about how users interact with 2FA and this informs improvements in our product. Because the complexity of 2FA is abstracted away from our customer’s integrations, it means we can make improvements to our service and often not require any changes to their code. Some customers have integrated with Authy and not made a change in years, yet during that time we’ve made many improvements to our service.

Not only is our API very easy to use, but we’ve also built the best 2FA smartphone app on the market. We are the highest rated on both the Apple and Google app stores and there are many blogs advocating the use of our end user application, including an “Excellent” rating from PC Magazine. It supports all the Authy 2FA methods, including push notifications but also works with applications that use the 2FA standard. Because of this, over 3 million users have downloaded our free app on mobiles and desktops for use with Google, Microsoft, Amazon, Facebook and hundreds of other services. The app can backup all the users’ accounts and allows them to be used across multiple devices. If someone loses a phone, no worries, they just restore their tokens on another device.

Don’t want to use our application? Already have a smartphone app? No worries, Authy has an SDK where you can embed all this functionality directly into your existing application with the minimum of development.

Let’s look at each area of building your own 2FA from the ground up and compare with using Authy.
<table>
<thead>
<tr>
<th><strong>Build It Yourself Challenges</strong></th>
<th><strong>Cost and Risk Factors</strong></th>
<th><strong>Buying Authy Advantage</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>DEVELOPMENT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implement your own solution based on the TOTP RFC standard</td>
<td>Risk of user abandonment if lacking support for all users, and device types, e.g., SMS cost/support</td>
<td>Faster time to deployment due to reduced development effort</td>
</tr>
<tr>
<td>Provide a simple user experience (UX) on par with current consumer apps</td>
<td>Cost - Initial dev average is 100k+ and the continuous maintenance, admin, hosting, security costs is around $100k+ per year</td>
<td>90% of the 2FA solution is maintained by Authy</td>
</tr>
<tr>
<td>Build, secure and maintain all back-end services (creation/protection onboarding, user/device lifecycle support)</td>
<td>Risk/Challenge – To build and deliver infrastructure capable of addressing telecom and network challenges in regional markets while maintaining global viability is both complex and expensive</td>
<td>Built in comprehensive functionality across all common 2FA methods (SMS, Voice, Offline, Push)</td>
</tr>
<tr>
<td>Build out admin, policy controls, logging, reporting, user/customer support tooling</td>
<td>Risk/Cost – Incomplete functionality could improperly lock out users</td>
<td>3rd party validation for deployment and maintenance</td>
</tr>
<tr>
<td>Provide timely support of all devices (laptops, PCs, phones, tablets, wearables) and operating systems (iOS, Android, OSX, Win)</td>
<td></td>
<td><strong>Authy Unique Attributes</strong></td>
</tr>
<tr>
<td></td>
<td>Most highly rated 2FA app in both Google Play and iTunes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Supports all devices (laptops, PCs, phones, tablets, wearables)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>End user self service with backup and multi-device capabilities</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Multiple service providers for SMS with intelligent logic to prefer certain providers in certain geographic regions</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPERATIONS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Build and maintain services for global highly reliable 24x7x365 uptime/availability/performance</td>
<td>2FA service interruption means locked out users, loss of use, loss of revenue, loss of reputation, fraud, high cost customer service and potentially lost customers</td>
<td>Redundancy and availability of service with lower costs</td>
</tr>
<tr>
<td>Staff to manage infrastructure and services</td>
<td>Challenging to deliver continuous high performance experience without erosion of UX</td>
<td>Lower cost of operations due to Authy’s geographically distributed support team</td>
</tr>
<tr>
<td></td>
<td>Cost - Staff for continual maintenance of servers, processes and upgraded functionality</td>
<td><strong>Authy Unique Attributes</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>99.999% uptime with no interruptions since launching in 2012 (verified status.authy.com)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;400 milliseconds response times despite continued growth of users on service</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Provides dedicated specialists in all areas - admin, ops, security, support</td>
</tr>
</tbody>
</table>
## SUPPORT

<table>
<thead>
<tr>
<th>Build It Yourself Challenges</th>
<th>Cost and Risk Factors</th>
<th>Buying Authy Advantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deliver functionality and staff a support team for 2FA end users.</td>
<td>If UX is challenging, onboarding requires additional customer support. Staff must be skilled/trained on all latest devices including set-up, security and usage. Must assist users locked out due to delivery challenges or because of lost or misplaced devices (creates social engineering security risk).</td>
<td>Can add or plug existing APIs into popular support desk solutions such as ZenDesk. Dedicated, continually trained support professionals.</td>
</tr>
<tr>
<td>Initial setup and tutorials for all users/customers</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Support for all user devices and upgrades | | **Authy Unique Attributes**
- Cloud backup and self-service helping users update to new devices without support assistance (minimizes risk of social engineering attacks)
- Minimal user training
- Built-in service redundancy coupled with user receipt verification and intelligent routing to ensure message delivery
- Built-in multi-device support, decreasing likelihood of lock out for lost/misplaced devices
- Skilled 24x7x365 support team with code level knowledge spanning all devices |
### Build It Yourself Challenges

**SECURITY**

- Authentication is heavily targeted (e.g., password, MITM, phishing attacks)
- Requires continuous, proactive detection and protection against constantly changing threats, malware, attacks and fraud
- Conduct routine vulnerability and penetration testing of entire service
- Protect data in transit and at rest
- Recover safely from potential breach
- Security must meet compliance standards

<table>
<thead>
<tr>
<th>Build It Yourself Challenges</th>
<th>Cost and Risk Factors</th>
<th>Buying Authy Advantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication is the front door - a breach here puts entire app or service at risk and potential take down</td>
<td>Attack, threat, fraud vectors are constantly changing and require full time attention, prevention and remediation</td>
<td>Purpose-built security for the cloud at every tier</td>
</tr>
<tr>
<td>Replacing breached components (e.g., user keys) often requires re-enrollment which takes time and can result in loss of user trust</td>
<td>Design and execute regular 3rd party testing and validation</td>
<td>All sensitive data encrypted and key data offline</td>
</tr>
</tbody>
</table>

**Authy Unique Attributes**

- Integrated & automated key rotation allows full silent key replacement with no user involvement should keys be compromised
- Regular 3rd party vulnerability testing
- 3+ years of cumulative production experience with no outages/interruptions

### SUMMARY

It’s clear from the evidence that building your own 2FA solution isn’t the right way to go. While it might seem attractive to put together a simple TOTP library and deliver tokens via SMS, the broader picture of supporting, maintaining and securing such features in your application is expensive.

Customers like Coinbase, Cloudflare, Twitch, Mercado Libre and SendGrid have chosen Authy to secure their customers accounts. Dell and VMWare also use Authy in their own identity management platforms. These customers looked at the value of Authy over building out all the infrastructure and developing all the code themselves. Choosing Authy was driven by the same reasons they didn’t build their own database nor web servers on which they build their applications.

Today we build on platforms and frameworks. We buy consumable services with the right level of functionality that are delivered through modern internet APIs. Authy is the right choice for enabling stronger security for your application and your customers data.

https://www.authy.com
sales@authy.com

@authy facebook.com/authysec linkedin.com/company/authy