CHECK OR MATE?
Strategic Privacy by Design

iapp

Most current approaches to "privacy by design" are mere Band-Aids, after-the-fact applications of either security controls or notice/choice controls. Proper PbD, embodying the principles of "privacy embedded into the design," "privacy by default," and being "proactive, not reactive," require more analysis, substantially more forethought, and design hinged on the privacy risks rather than current practices.

In chess, thinking ahead is critical for success. For skilled players, the game is about strategic thinking, not simply calculating the best outcome a few moves ahead. If your opponent is thinking strategically and you're just trying to capture the next piece, you may win the battle but lose the war.

Strategic thinking is also critically important in designing products and services that respect privacy and ensure individual trust. If all you're trying to do is address one issue or institute one control, you may succeed at mitigating that issue, but ultimately, you may lose the war against an onslaught of potential privacy risks.

In my observation, most organizations' PbD programs rely heavily on privacy impact assessments. I contend that most PIAs are principally reactive, not proactive, for two reasons. First, PIAs generally come after a design has been proffered by the engineering team, often foregoing potentially privacy-friendly design architectures. Secondly, PIAs, often compliance driven, frequently couch privacy risks in terms of an absence of controls (failure to inform data subject, lack of security controls, lack of data deletion, etcetera), not in terms of addressing actual underlying privacy risks.

Finally, the bifurcated approach — first, engineering staff with their initial design, and second, the privacy analysts with their PIAs — reinforces biases that exist within these groups. The engineers or those in IT security tend to focus almost exclusively on encryption and access control as privacy controls, viewing privacy as a confidentiality issue in the CIA triad (confidentiality, integrity, availability).

On the other side, mirroring the historic regulatory presence of the Fair Information Principles, a company's privacy analysts and lawyers tend to focus on notice and choice as a control mechanism, with an acknowledging nod to encryption and access controls as necessary technical security controls. My anecdotal observation of corporate behavior is supported by a recent study by Ari Waldman, soon to be published in Houston Law Review, which similarly found a limited set of controls being applied. "But, for the most part," Waldman writes, "technologists and firm lawyers thought about privacy in narrow ways, either as synonymous with encryption or limited to notice-and-choice."

It is also interesting to note that these two categories of controls are supplementary. In other words, they can be added on top of any existing system but give little to privacy forethought in the underlying design of the system. To analogize to the chess game, this is like looking at a chess board and focusing only on your next move: preventing a key piece from being taken (using encryption to defend against an attacker) or attacking an opponent's piece before their next move (putting a data subject on notice in advance of using or sharing their information), with nary a consideration to long-term perspective. In making privacy aforethought, designers must consider a different way of achieving better and more strategic "game-winning" results — different data flows and different business models, which still achieve the objective.

What follows is my attempt to contrast the two approaches, the all-too-common PIA-based PbD approach and the proactive — or what I'm calling strategic — PbD approach that I take and teach (see, for instance, the IAPP webinar Embedding Privacy by Design for an early and shorten variant of this). Obviously, in the space of this paper, I can't detail the approaches in their entirety, but I'll try to focus on the salient points.

The Scenario

Consider a hypothetical game developer at a company who wants to develop a chess game for users to play each other via their mobile devices. Because chess players are of such varying skill levels, it's appropriate to have some matching mechanism to coordinate games from players of similar skill level. Failure to do so would result in advanced players getting bored and less-skilled players getting frustrated.

For the purposes of simplification in this illustration, I'm going to forgo all the whiz-bang things a developer may include as requirements. All those could be dealt with, in turn, if they were part of the requirements. For this analysis, I'll focus just on the most basic use case using the requirements identified above.

Approach 1: Privacy Impact Assessment

In a very common scenario, probably familiar to most in the privacy space, the developer sets out creating the application, absent any privacy considerations. In the example, the hypothetical developer decides he will use a basic client-server model and needs three tables in a server database:

PLAYERS
ID
Username
Password
SkillLevel

A **Players** table with an auto-incrementing primary key ID, a user name determined by the user, their password to validate their account, and the user's current skill level.

GAME QUEUE
PlayerOneID
ID

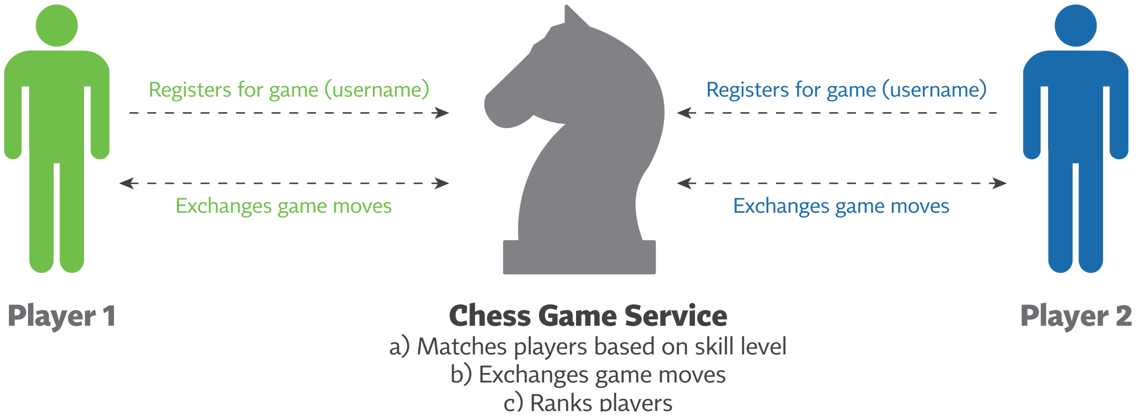
A **Game Queue** table to keep a list of players awaiting similarly skilled players to a match. This table is linked to the **Players** table through PlayerOneID (which is a foreign key to the Players ID as a primary key), so that when another player indicates a willingness to join a game, they can be matched against an existing player of the same skill level in the queue through the **Players** table.

GAMES
ID
Date-Time
PlayerOneID
PlayerOneSkillLevel
PlayerTwoID
PlayerTwoSkillLevel
WhoseTurn
LastMove
Winner

Finally, a **Games** table that holds the two players' IDs (foreign keys linked to the **Players** table) and skill levels (at the time of game play). It also lists whose turn it is and what the last move was (to report to the other player). Once a winner is declared, that is kept in the table, as well. The game's result is used to adjust the players' current skill level for future matches.

Note: I assume the database isn't keeping historical moves, but one could also likely see a design employing a moves table to keep a history of all moves in the game.

Basic Information Flow



The developer has the basic design and considers some security protections. The developer decides that, using best practices, the player's password should be hashed and makes sure that when a player logs in to the app, the communications are over HTTPS to prevent the password being sent in cleartext. Seeing no other security considerations (refrain: "Hey, it's not personal information, and there is no financial data"), the developer sends his design over to the company's privacy analyst for a review. The analyst does a cursory review and, even though the game is collecting minimal personal data, decides to conduct a PIA. Based on the PIA, she identifies a few risks and mitigating solutions (representative sample not meant to be exhaustive):

Risk	Solution
Username might reveal information to other players (i.e. "johnsmith101283").	Player should have the choice to appear as anonymous to other players.
Player might not understand how information is being collected, used or shared.	Create a privacy statement, and publish it to the players when they start the app.
Right of erasure	Players should be able to delete their accounts.
Data retention	Game data should be deleted after no longer necessary to fulfill the purpose for which it exist (game play and skill-level assessment).

As a result, the developer goes back and adds in the solutions the analyst has proposed:

1. An anonymous flag is added to the **Players** table, and player's username will be displayed as "Anonymous" to the other player during a game if they have this flag set to true.

2. A link to the company privacy notice is put into the app.

3. A delete-account option is added to the app that deletes the player from the Players table.

4. After 30 days, games are deleted from the Games table.

Voila! We're done, right? Privacy by design! Our game has been designed with privacy in mind.

Au contraire, mon frère!

Notice first that all except the first "risk" are described in terms of those unrealized controls (lack of notice, lack of choice in deletion, lack of security control). The reason for this is because many controls will mitigate multiple privacy risks. For instance, data retention and right of erasure help alleviate future risks of insecurity in the system, identification of individuals, aggregation of data with outside data sets, and so on. However, these aren't the only controls available, and using them reduces the possible design universe and use of other controls.

Second, we've only addressed a bare minimum of the privacy risks. In competitive chess, being able to know the types of moves an opponent is likely to make is important for developing counter strategies. We've haven't prevented the chess game service provider from keeping a history of people's moves, which could be used by an employee or a hacker to beat the person in a competitive match (potentially the anonymity flag would prevent player opponents from developing such a dossier, but players are limited to people they play, whereas employees or hackers could get the entire universe of player data). We haven't prevented someone (game company employee, hacker or the service itself, etcetera) with access to the database from distorting a player's skill level. The design doesn't prevent reidentification of the player based on username, since many people use the same username elsewhere. Perhaps, as a follow through, that reidentification can be used to exclude people from the system ("No Russian chess players allowed!").

The fact is dozens of other potential privacy issues are not mitigated.

> ***The bare essence of PbD is that the original design should consider privacy, not be a hodgepodge of miscellaneous solutions "bolted on"***

The bare essence of PbD is that the original design should consider privacy, not be a hodgepodge of miscellaneous solutions "bolted on," as Ann Cavoukian is prone to say, to address a few identified risks. As with chess, in PbD, you must plan your strategy ahead of time, not be reactive. Strategic game play involves positioning your pieces, in advance, on the board to resist an attack, not reacting to your opponent calling "check" by moving a pawn in between them and your king.

Approach 2: Strategic Privacy by Design[1]

In the proactive approach, the developer needs to describe the use case, identifying the actors and the functionality of the system. For this chess game, there are five relevant actors:

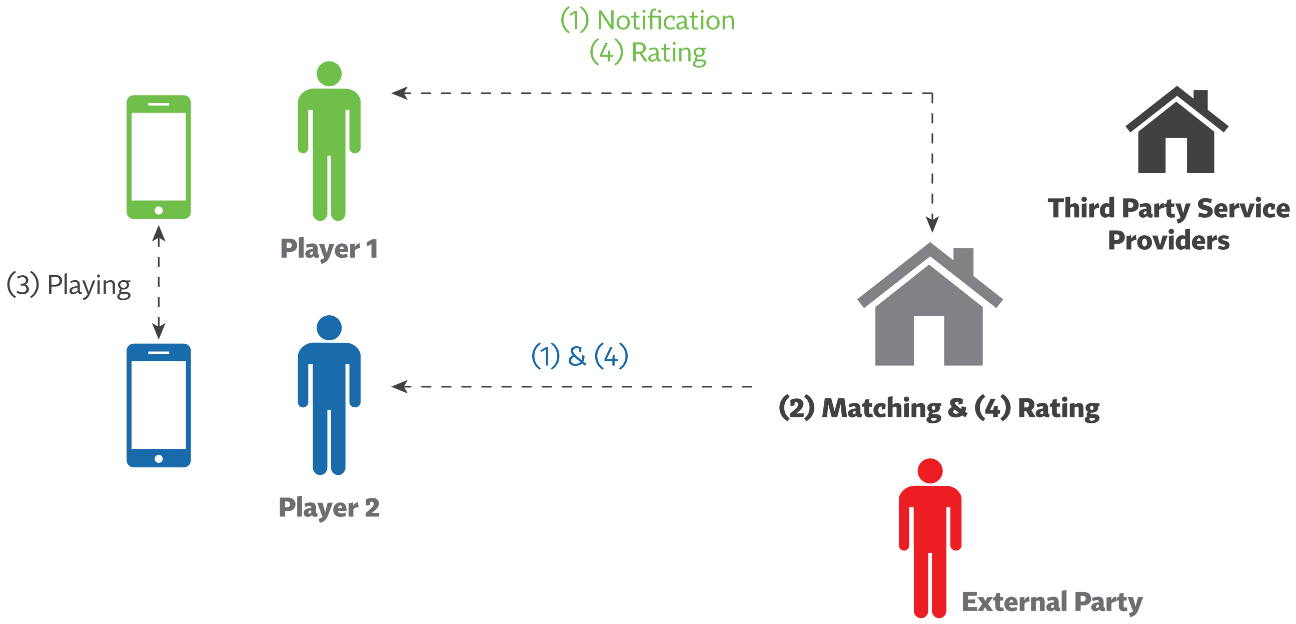
- Players
- Game developer (presumably us)
- Third-party service providers (telecom, hosting, mobile phone OS providers, etcetera)
- Our employees
- External parties (hackers, government, our employees, etcetera)

The basic use case consists of (1.) a player notifying the game developer that they want to play a game; (2.) a game developer matching players based on skill level; (3.) the players playing a game of chess; and (4.) a game developer using the results of that game to determine the skill level for future games. Now at this point, we haven't at all defined how this is going to work, how we match players, how the game is played between the players, how data flows, etcetera. We're just defining a simple description of the use case.

The next step is for the game developer, with assistance perhaps from a privacy analyst, to scope the privacy requirements. I use Dan Solove's "Taxonomy of Privacy" as

[1] **Note:** What I'm calling "Strategic Privacy by Design" is a lightweight version of privacy engineering. While the process is like what one might undertake in a formal privacy engineering analysis, this foregoes much of the formality of the systems engineering approach, making it much more useful to a wider audience.

Environment



a model because I find it pinpoints all the relevant privacy issues that could crop up and provides a very systematic approach to stepping through the potential adverse events. Others may use Ryan Calo's "Privacy Harms model," Nissenbaum's "Contextual Integrity," or a compliance-based model, such as the GDPR.

However, my take is these other risk models require the analyst applying the model to contrive more specific risks in a non-systemic way. In the case of compliance model, they are usually control based not risk based, as discussed above. CIPTs among the readers will note the CIPT textbook remarks that none of the risk model frameworks are sufficient in and of themselves because they target different elements in the risk model (threats, vulnerabilities or events), and a combination of models may be better.

For most applications, it is also necessary to consider all the different individuals whose privacy might be affected. It could be system users (players, in our use case). It could be employees. It could be bystanders. There are often several individuals in each use case whose privacy could be adversely impacted. For our case, though, the only relevant individual is the player. It's also important to think of each of the actors as a potential privacy invader. A full analysis is illustrated below showing 53 identified risks.

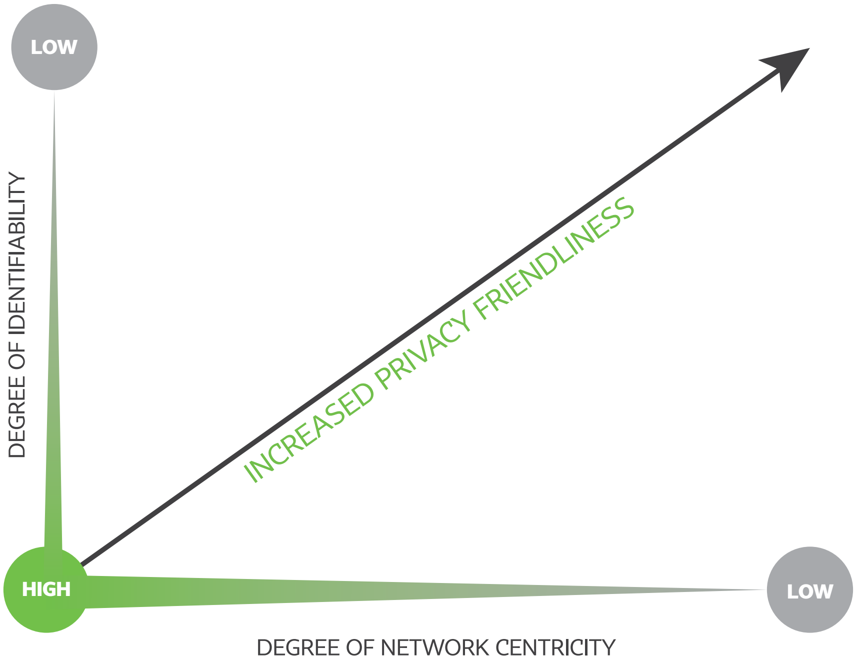
The third step is to choose an architecture. For that, I turn to "Engineering Privacy" by Sarah Spiekermann and Lorrie Cranor, which points out the less identifiable and

Table 1: Privacy risks pre-architecture, showing 53 identified risks.

CATEGORY	PRIVACY INVASION	INITIAL ASSESSMENT				
		Other Players	Game Provider	Third Parties	Inside Party	External Party
Information Collection	Surveillance		X	X	X	X
	Interrogation	X	X			
Information Processing	Aggregation	X	X	X	X	X
	Identification	X	X	X	X	X
	Insecurity	X	X	X	X	X
	Secondary Use	X	X	X		
	Exclusion	X	X	X	X	X
Information Dissemination	Breach of Confidentiality		X		X	X
	Disclosure	X	X	X	X	X
	Exposure					
	Increased Accessibility		X			
	Blackmail	X	X		X	X
	Appropriation	X	X		X	X
	Distortion	X	X	X	X	
Invasion	Intrusion	X	X	X		
	Decisional Interference					

less centralized the data, the higher degree of privacy friendliness. See the charts below:

The developer, deciding that because the company's leadership has made privacy a priority, will prioritize privacy friendliness and design a medium level of centralization and low level of identifiability. Further, they have no other reason to choose otherwise as the use case (a chess app) allows for such a privacy-friendly approach. There may be reasons that a company needs a highly centric, highly identified system, but too often this is the default out of habit and not careful evaluation.


LOW
DEGREE OF IDENTIFIABILITY
INCREASED PRIVACY FRIENDLINESS
HIGH
LOW
DEGREE OF NETWORK CENTRICITY

Identifiability	Linkability	System Characteristics
Anonymous	Unlinkable	• No collection of contact information • No collection of long-term personal characteristics • k-anonymity with large value of k
Pseudonymous	Not linkable with reasonable effort	• No unique identifiers across databases • No common attributes across databases • Random identifiers • Contact information stored separately from profile or transaction information • Collection of long-term person characteristics on a low level of granularity • Technical enforced deletion of profile details at regular intervals
	Linkable with reasonable & automatable effort	• No unique identifiers across databases • Common attributes across databases • Contact information stored separately from profile or transaction information
Identified	Linked	• Unique identifiers across databases • Contact information stored with profile information

Based on that architectural choice of low identifiability, the developer of the chess app needs to design a system with the following characteristics:

1. No collection of contact information

2. No collection of long-term personal characteristics

3. k-anonymity with large value of k

Most developers at this point are probably scratching their heads saying, “No way! You can’t build a system like this, that meets these requirements.” Are we so easy to give up at this point? No, and I’m going to offer a solution that meets the needs and still accomplishes the objective of the use case.

Player A generates a one-time public/private key pair (new for each game), then submits the public key to the server, along with a recent history of wins/losses (more about that below). Based on that recent history, the server makes a spot determination of skill level and puts the player in a queue. Player A’s app is constantly asking the server, “Have you found a match for me?”

Player B does the same and is matched by the server based on skill level with Player A. Once matched, the server exchanges each player’s public key with their opponent.

Players A and B communicate chess moves through a message field, using digitally signed encrypted messages of moves to each other. Once the game concludes, the loser signs a message conceding to the winner.[2] The winner can submit that to the server and receive a digitally signed statement that says, assuming he was playing a chess master, “You beat a master at <DateTime>.” The loser submits their concession and receives a similar statement of loss. This is what constitutes the history of wins/losses referenced above.

Below is the server database table for this design:

GAME
PlayerOnePublicKey
SkillLevel
PlayerTwoPublicKey
Message

A **Game** table to keep a list of players awaiting similarly skilled players to match. When a second player requests a game, the player’s public keys are exchanged. They can then exchange moves via a generic encrypted message field. Once a winner is announced (or a draw), the record is deleted.

As you can see, we’re meeting part of our architectural goals. The system has a low level of identifiability. There is no collection of long-term characteristics. There is no contact information collected. Finally, if skill level has only some rough measures (beginner, novice, amateur, master, grand master), then they are sufficiently large sets and low level of granularity, aka k-anonymity with large k. We’ve partially decentralized because some of the data is now stored locally in the app (such as history of wins and losses and the unencrypted moves for the games).

I would submit that the new design addresses many, though not all, of the privacy issues. For that we turn to some additional controls. By example, an employee could still distort a person’s skill level in the database, though this is partially mitigated by the employee not knowing whom they are distorting. Further decentralization would also reduce the

[2] You may be asking why would the loser admit to losing? It’s true that someone could modify the app to prevent admitting defeat. Ultimately, they are only harming themselves because they may be ranked higher than appropriate and will face players they can’t beat. If we felt this was a serious problem, we could build in additional integrity checks, such as submitting disputed games to an arbiter.

ability of an employee to distort random gamers (see sidebar on page 10). For additional controls, we can use traditional integrity mechanisms, such as access controls, logging and auditing.

Now you may be asking at this point a few questions. First, you may be thinking of additional functionally that might be important: How is the game developer making money? What if we want a leaderboard or to offer prizes to top players? What if want to facilitate playing games with your Facebook friends?

These are legitimate questions, and this goes to properly defining the use case upfront. Had these been requirements from the start, the design may turn out different. This points to a critical failure in most design, the failure to sufficiently detail the use case.

Another option is to build this most core use case in the most privacy-friendly fashion and then layer additional functionality on top. That way, if the additional functionality has some specific privacy implication, it is isolated, potentially optional (think about FIPP's Choice/ Control) for the player, and a privacy risk analysis can be focused on just that component without disturbing the baseline analysis.

Conclusion

When presenting such an approach, I often receive these sorts of comments, to which I respond below:

Table 2: Privacy risks post-architecture showing 34 risks fully or partially mitigated.

CATEGORY	PRIVACY INVASION	INITIAL ASSESSMENT				
		Other Players	Game Provider	Third Parties	Inside Party	External Party
Information Collection	Surveillance		!	✓	✓	!
	Interrogation	✓	X			
Information Processing	Aggregation	✓	✓	X	✓	✓
	Identification	✓	✓	X	✓	✓
	Insecurity	X	✓	X	✓	✓
	Secondary Use	✓	!	X		
	Exclusion	✓	✓	✓	✓	✓
Information Dissemination	Breach of Confidentiality		✓			
	Disclosure	!	!	!	!	!
	Exposure					
	Increased Accessibility		✓			
	Blackmail	✓	✓		✓	✓
	Appropriation	✓	✓		✓	✓
	Distortion	!	!	✓	!	
Invasion	Intrusion	✓	X	X		
	Decisional Interference					

Decentralization

We could achieve a much lower degree of centralization and additional risk mitigation by handing off game play to the players to interact directly, where they only report the results to the central authority for imprimatur. Naively, this would require providing direct contact information between parties, something that would impact our identifiability issue. Additional measures, such as using random XMPP servers or having the app function as a Tor hidden service to exchange moves, would alleviate this.

Full decentralization (i.e., no central authority for matching players, certifying game results or determining expertise) is achievable but requires a much higher degree of forethought, employing mechanism design and techniques like other fully decentralized systems, such as bitcoin and BitTorrent.

"This is hard."
New concepts are often viewed as difficult to understand. Building systems with a privacy-friendly approach can seem foreign, especially to developers with a traditional mindset. One problem that does crop up, however, is that many developer tools are not natively capable of privacy-preserving functionality. Consider relational databases. Primary and foreign keys maintain relationships between records. But maybe I don't need bi-directional knowledge. I need to know that Jane Doe checked out "Lady Chatterley's Lover" from the library, but I don't want to be able to produce a list of all library patrons, who checked that book out, upon receipt of a subpoena. Yes, this can be done (for instance, using trapdoor functions and isolated data processing), but it isn't the native way relational databases work nor the mindset of most developers.

"This is a special case; my application can't be built this way."
Without doing the analysis, it's hard to say. I will agree that some applications may have special use cases, but I would argue that many more could use this this level of scrutiny and design than are subject to it. The problem is most staff hasn't been trained, and there isn't sufficient emphasis put on privacy and trust as a design requirement. I have yet to meet an application that couldn't benefit from this type of approach, only designers too wedded to privacy-unfriendly architectures. Developers, as well as chess players, suffer from the Eistenllung effect — the cognitive bias that prevents people from finding superior solutions to problems where they have known solutions.

"This is extreme. We don't need to go to this degree to mitigate risks."
One aspect that this shortened analysis forgoes is assessing the impact and likelihood of the privacy issues as not all risks are created equal. But it isn't always the most common risks that make the news, like the church-home owners intruded upon by Pokemon Go players, the minorities excluded by Airbnb hosts, or the pregnant teenager targeted by Target.

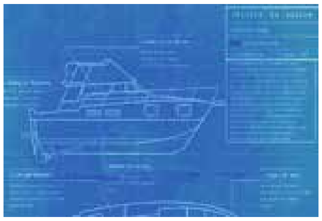
It is important to take this into your risk calculation.

Even so, a more tempered approach to the design might be available. In the architectural analysis outlined above, using pseudonymous identifiability (rather than fully anonymous), along with a more centralized design, allows for that middle ground.

The point of the comparison above is to illustrate a stark contrast between solutions that might be the result of a retroactive application of controls to a proposed design and a proactive design built with privacy in mind.

Table 3: Comparison of approaches based on PbD principles

Privacy by Design Principle	Approach 1—PIA	Approach 2—Strategic
Proactive not reactive	! While not reactive in the traditional sense (remediating breaches with credit monitoring), Approach 1 instills controls that are reactive to the proposed design rather that the privacy risks proactively guiding the initial design.	✓ In Approach 2, the potential risks of the use case are identified first. This then guides the creation of a privacy-friendly architecture and additional controls to mitigate further risks. Determining the controls necessary upfront suggests a design rather than the design limiting the available control choices.
Privacy as the default	✗ In Approach 1, players would need to take additional steps to mitigate some of the risks, such as using a random username, changing usernames, not reusing their passwords on another site, selecting the anonymous option, etcetera.	✓ In Approach 2, there isn't much else the player can do to mitigate additional risks. They could connect via Tor or use a VPN to prevent ISP snooping, but most of the risks are mitigated by the design.
Privacy embedded into the design	✗ Privacy-protecting controls could easily be removed; they aren't embedded in such a way that the system can't function without them. HTTPS could default to unencrypted HTTP, unless the server admin is using HSTS. The privacy statement could be unlinked or made inaccessible. The anonymous choice option provided to players could fail, revealing the player's identity.	✓ In Approach 2, the system won't function without the fundamental privacy controls. There simply isn't an opportunity for controls to fail (i.e., Player 1 can't send a move without the other player's public key as Player 2 will be looking for an encrypted move).
Full functionality	✓ Both approaches meet the desired requirements of a chess app that matches players by skill level.	
End-to-end security	✓ Both approaches secure data end to end (assuming all security controls are in place and aren't allowed to default to an insecure state).	
Visibility and transparency	! Approach 1 provides a privacy statement, but who reads privacy statements…on a mobile device?	✓ While not explicitly stated, Approach 2 would likely include a privacy statement as an additional control. It would primarily rely on in-app context. For instance, the limited request for player information and non-revelation of competitor's information provides contextual clues about the lack of information collected and shared. (While Approach 1 shows "You're playing ChessMaster99," Approach 2 might show "You're playing an EXPERT.")
Respect for user privacy	✓ Arguably, both approaches respect players, but Approach 2 is more comprehensive, mitigates more risks, and better addresses the other PbD principles.	



Jason Cronk is a privacy and trust consultant with Enterprivacy Consulting Group. He holds a JD, a CIPP/US, a CIPT, a CIPM, and is a FIP. His consulting practice is focused solely on the niche of privacy by design, where he helps companies build privacy-friendly services or trains others on how to follow this strategic PbD approach. Upcoming Strategic Privacy by Design workshops are posted at https://enterprivacy.com.